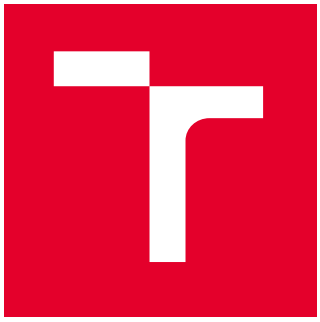


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

SYSTÉM PRO ROZPOZNÁNÍ HLASOVÝCH POVELŮ V REÁLNÉM ČASE

REAL-TIME VOICE COMMAND RECOGNITION SYSTEM

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE

AUTHOR

Evžen Šíbl

VEDOUCÍ PRÁCE

SUPERVISOR

Ing. Jiří Přinosil, Ph.D.

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Audio inženýrství**
specializace Zvuková produkce a nahrávání
Ústav telekomunikací

Student: Evžen Šíbl ***ID:*** 221456
Ročník: 3 ***Akademický rok:*** 2021/22

NÁZEV TÉMATU:

Systém pro rozpoznání hlasových povelů v reálném čase

POKYNY PRO VYPRACOVÁNÍ:

V rámci práce se seznamte s problematikou zpracování řečového signálu se zaměřením na rozpoznávání hlasových povelů. Na základě získaných znalostí navrhněte a implementujte algoritmus, který dokáže spolehlivě rozpoznat alespoň 10 hlasových povelů od různých osob i v mírně zarušeném prostředí. Algoritmus musí být navržen pro práci v reálném čase bez využití cloudových služeb. Experimentálně ověřte použití metod pro potlačení hluku okolního prostředí na dosažené výsledky a stanovte možnosti jejich použití.

DOPORUČENÁ LITERATURA:

[1] DE ANDRADE, Douglas Coimbra, et al. A neural attention model for speech command recognition. arXiv preprint arXiv:1808.08929, 2018.
[2] AZARANG, Arian; HANSEN, John; KEHTARNAVAZ, Nasser. Combining data augmentations for CNN-based voice command recognition. In: 2019 12th International Conference on Human System Interaction (HSI). IEEE, 2019. p. 17-21.

Termín zadání: 7.2.2022 ***Termín odevzdání:*** 31.5.2022

Vedoucí práce: Ing. Jiří Přinosil, Ph.D.

doc. Ing. Jiří Schimmel, Ph.D.
předseda rady studijního programu

ABSTRAKT

Bakalářská práce se zabývá tvorbou systému pro rozpoznání hlasových povelů. Klasifikátor tohoto systému byl vytvořený pomocí neuronové sítě. V práci se obeznámíte s historií a problematiku rozpoznání řeči. Byl vytvořený systém, který detekuje v nahrávce úsek obsahující řečový signál, který následně pomocí klasifikátoru rozhodne o jaké slovo z tabulky slov se jedná. Byly vytvořeny 3 modely se stejnou architekturou avšak s různými trénovacími daty. Tyto modely byly následně porovnány mezi sebou. Pro výsledný systém bylo vytvořené jednoduché uživatelské rozhraní.

KLÍČOVÁ SLOVA

neuronové sítě, rozpoznání řeči, mfcc, dataset, VAD, Keras API

ABSTRACT

The bachelor thesis deals with the development of a system for voice command recognition. The classifier of this system was created using a neural network. In this thesis you will learn about the history and problems of speech recognition. A system has been created that detects a section in a recording containing a speech signal, which then uses the classifier to decide what word from the word table it is. Three models with the same architecture but with different training data were created. These models were then compared with each other. A simple user interface was created for the resulting system.

KEYWORDS

neural networks, speech recognition, mfcc, dataset, VAD, Keras API

ŠÍBL, Evžen. *Systém pro rozpoznání hlasových povelů v reálném čase*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2021, 50 s. Bakalářská práce. Vedoucí práce: Ing. Jiří Přinosil, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Evžen Šíbl
VUT ID autora:	221456
Typ práce:	Bakalářská práce
Akademický rok:	2021/22
Téma závěrečné práce:	Systém pro rozpoznání hlasových povelů v reálném čase

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Jiřímu Přinosilovi, Ph.D. za odborné vedení, konzultace, rychlou komunikaci, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Cílem této práce je návrh a sestavení systému pro rozpoznání hlasových povelů. Zároveň práce obsahuje stručnou historii této problematiky a základní vysvětlení různých přístupů rozpoznání řeči v signálu.

V teoretické části této práce se seznámíme s různými metodami zpracování této problematiky – od jednoduchých analýz zvuku přes komplexní algoritmy zpracování signálů až po strojové učení. Také je zde obsažena problematika detekování řeči v nahrávce, neboli Voice Activity Detection (zkráceně VAD). Obeznámíme se také s metodami pro odstranění nežádoucích složek ze signálu.

V praktické části této práce se nachází návrh, kontrukce, testování a implementace systému pro rozpoznání hlasových povelů. Návrh obsahuje samotný návrh celkového systému, ale také obecný návrh klasifikátoru, který je realizován pomocí konvoluční neuronové sítě.

Neuronová síť je vytvořena pomocí Keras API v programovacím jazyce python. Jako počáteční trénovací data byly použity nahrávky z volně dostupné databáze izolovaných jednosekundových slov od společnosti Google. V sekci konstrukce je také popsán způsob tvorby použitelných datasetů a jejich následné zpracování. Pro zlepšení přesnosti klasifikátoru je využita knihovna `keras-tunner`, která pomáhá ulehčit proces hledání vhodných hyperparametrů pro neuronovou síť.

Po vytvoření architektury neuronové sítě, která dosahovala adekvátních výsledků, je síť použita na natrénování 3 různých modelů založených na 3 různých datasetů. Mezi tyto datasety patří dataset obsahující již zmíněné prvotní nahrávky (nahrávky převážně neobsahovaly ruch), dataset složený z nahrávek obsahující ruchovou část a dataset složený z nahrávek, ve kterých pomocí externí funkce jsou odstraněny ruchové prvky. Vytvořené modely jsou porovnány mezi sebou.

Také se v práci nachází řešení algoritmu VAD.

Všechny tyto dílčí části jsou propojeny pomocí jednoduchého uživatelského rozhraní vytvořeného pomocí knihovny `PySimpleGUI`.

1 Rozpoznání řeči

1.1 Historie

Už od počátku elektronických strojů bylo rozpoznání řeči ve společnosti diskutovaným tématem. Lidé chtěli ovládat své věci, nástroje pomocí hlasu. To však ve své době nebylo příliš možné až do příchodu moderního počítače.

První zaznamenané rozpoznání hlasu byla hračka Radio Rex z roku 1922. Jednalo se o hračku, která vystřelila psa ze své boudy, když člověk v její blízkosti vyslovil slovo „Rex“. Fungovala na takovém principu, že vyslovení slova „Rex“ dosahovalo podobné akustické energie, která byla potřeba k rozpojení obvodu a uvolnění pružiny, která držela psa v boudě [1].

V roce 1952 byl vynalezen první stroj, který se naučil naslouchat. Jednalo se o jedno ciferný rozpoznávač od společnosti Bell Labs. Přístroj dokázal rozeznat vyslovení čísel 0 až 9 od jedné osoby [1]. Tato čísla byl schopen rozpoznat pomocí zprůměrování akustické energie takzvaných formantů – špičky ve zvukovém spektru, které vznikají pomocí rezonance v hlasovém ústrojí a rozpoznávají se pomocí nich jednotlivé samohlásky.

Pár let na to v roce 1957 vznikl ve společnosti RCA Laboratories první rozpoznávač deseti slabik [1]. Je důležité zmínit, že systém byl vytvořen pro rozpoznání opět pouze jednoho referenta.

Tuto problematiku vyřešili na MIT, kdy ve stejné době vytvořili systém pro rozpoznání deseti samohlásek, který nebyl závislý pouze na jednom mluvčím.

Od konce šedesátých let začal technologický trh exponencionálně růst s příchodem MOSFET tranzistorů. Technologické pokroky vznikaly denně a firmy se začaly předbíhat ve svých produktech. Jeden z těchto „závodů“ dal vzniknout v roce 1987 první plně interaktivní hračce. Jednalo se o panenku Julii od firmy Worlds of Wonder, která byla schopna reagovat na vámi vyřčená slova a pomocí syntézy hlasivek i odpovídat [1].

Největší pokrok přišel v roce 1990, kdy Dragon Systems vytvořili první *speech to text* systém s velkou slovní zásobou. Dragon dictate, jak se tento systém jmenoval, byl přelomový, avšak řečník musel mluvit po jednotlivých slovech. To se změnilo roku 1997 kdy Dragon Systems představili systém pro rozpoznání souvislé řeči. Uživatel mohl mluvit přirozeně a nemusel dělat pauzy mezi slovy. Oba tyto systémy byly založeny na skrytém Markovovém modelu (Hidden Markov model – HMM). Byla zde využívána statistika na předpovídání jednotlivých slov, frází a vět. Díky tomu bylo možné pochopit kontext vyslovené věty a určit, které slovo pravděpodobně bylo vyřčeno oproti obdobně znějícím slovům, jelikož systém byl schopen pochopit jeho význam.

Tyto pokroky byly klíčové ke zdokonalení metod rozpoznání řeči. Všechny metody, které vznikly do současnosti, jsou následující [1]:

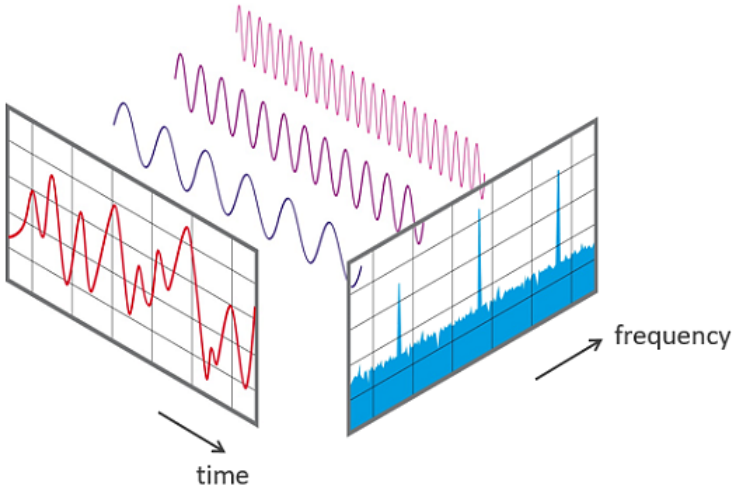
- Spektrální analýza – FFT
- Cepstrum
- Dynamická časová deformace – DTW
- Skryté Markovovy modely – HMMs
- Jazykové modely
- Strojové učení a neuronové sítě

1.2 Metody rozpoznání řeči

1.2.1 Spektrální analýza

Jedná se o analýzu z hlediska spektra frekvencí, nebo jiných veličin, jako je například energie, vlastní čísla apod. Z hlediska audia se využívá Rychlá Fourierova transformace (Fast Fourier transform – FFT).

Diskrétní Fourierova transformace (Discrete Fourier transform – DFT) převádí audio signál na jednotlivé spektrální složky, a tím poskytuje frekvenční informace o signálu (obr. 1.1 převzato z [2]). Fourierova transformace je transformace funkce času na frekvenci. Rozdíl mezi DFT a FFT je ten, že FFT tuto transformaci dokáže provést mnohem rychleji $O(NlogN)$ než DFT $O(N^2)$ [3].



Obr. 1.1: Zobrazení signálu v časové a frekvenční oblasti

Diskrétní Fourierova transformace je definovaná vzorcem

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} nk}, \quad k = 0, \ldots, N-1, \tag{1.1}$$

kde $x_0, \ldots, x_{N+1}$ jsou komplexní čísla.

Nejznámější FFT algoritmus *Cooley-Tukey algorithm* funguje na takzvaném principu „rozděl a panuj“. Algoritmus rekurzivně rozloží DFT o velikosti složeného čísla $N = N_1 N_2$ na mnohem menší DFTs o velikostech N_1 a N_2 [3].

Přístroje, které využívají FFT a spektrální analýzu, se nazývají spektrální analyzátory. V minulosti byl také používán analyzátor s kmitavým laděním (swept-tuned analyzer). Používal přijímač ke konverzi části spektra vstupního signálu na střední frekvenci úzkopásmového filtru, jehož okamžitý výstupní výkon byl zaznamenáván jako časová funkce. Právě pomocí tohoto *swept-tuned analyzatoru* vytvořila firma Bell Labs ve své době již zmíněný jedno ciferný rozpoznávač.

1.2.2 Cepstrum

Cepstrum je výsledek inverzní Fourierovy transformace logaritmu odhadovaného spektra signálu.

Rozlišujeme několik druhů:

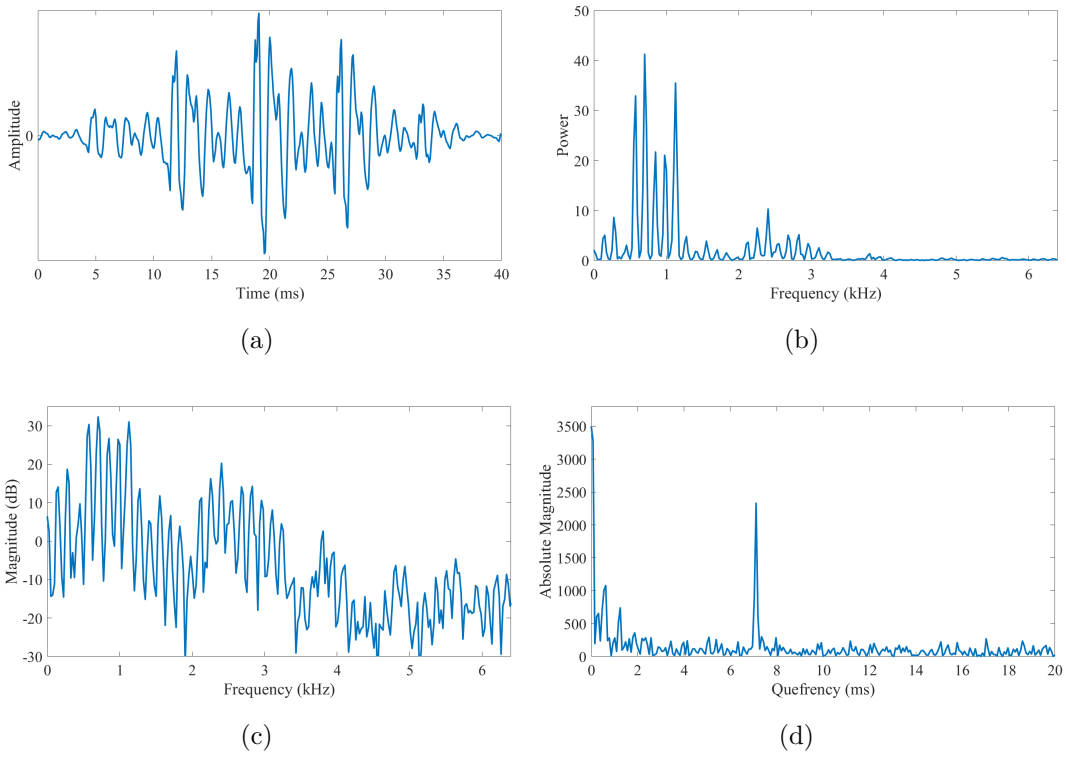
- komplexní cepstrum,
- reálné cepstrum,
- fázové cepstrum,
- výkonové cepstrum.

Nejčastěji se v oborech využívá výkonové cepstrum (power cepstrum) a oblast rozpoznání řeči není výjimkou. Využívá se tak moc oproti ostatním, že přebralo celkovou definici tohoto slova.

Cepstrum je v podstatě spektrum spektra. Matematicky zapsáno jako:

$$C(x(t)) = |F^{-1}[\log(|F[x(t)]|^2)]|^2. \tag{1.2}$$

Nejdříve se na zvukový signál (obr. 1.2(a)) použije DFT a z časové oblasti se přejde do frekvenční oblasti, a tím vznikne výkonové spektrum (obr. 1.2(b)). Poté se převede výkonové spektrum na logaritmické výkonové spektrum (obr. 1.2(c)). Osa výkonu se převede do logaritmické metriky. Na závěr se aplikuje na toto spektrum inverzní Fourierova transformace, jelikož se může zacházet s logaritmickým výkonovým spektrem jako s audio signálem. Tímto se vytvoří cepstrum (obr. 1.2(d)) a osa x se přehodí na tzv. quefrency (jedná se o časové měřítko, ale ne ve smyslu signálu v časové oblasti). Špička, která se nachází v grafu cepstra (kolem 7 ms) se nazývá 1. rhamonická, ta znázorňuje přítomnost základní frekvence. Tento vrchol se v cepstru vyskytuje proto, že harmonické složky ve spektru jsou periodické a perioda odpovídá základní frekvenci, protože harmonické jsou celočíselnými násobky základní frekvence. Kdybychom vyhladili Log-výkonové spektrum, zjistíme,



Obr. 1.2: (a) Zvukový signál, (b) Výkonové spektrum, (c) Log-výkonové spektrum, (d) Cepstrum

že jeho špičky odpovídají jednotlivým formantům v signálu. Formanty jsou výsledkem akustické rezonance lidského hlasového ústrojí a pomocí nich se dá rozpoznat, jaké písmeno bylo řečeno.[1]

Mel-frekvenční cepstrum (MFC) znázorňuje velmi krátké období ve výkonovém spektru založeném na lineární Kosinově transformaci Log-výkonového spektra, jejichž průběh je na nelineární mel stupnici frekvencí.

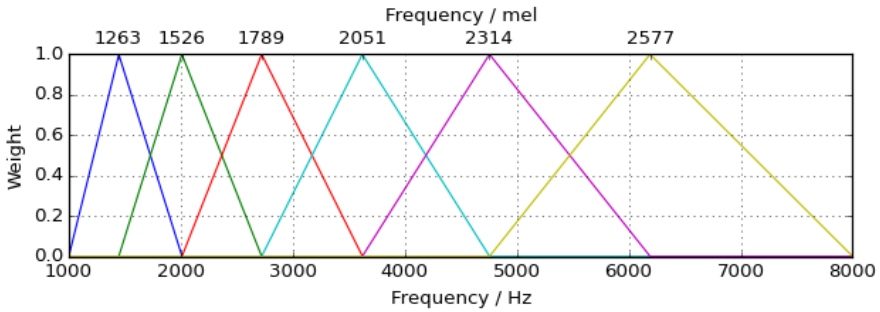
Mel-frekvenční cepstrální koeficienty (MFCC) jsou koeficienty, jenž tvoří MFC. Tyto koeficienty se odvozují z cepstra [5].

Rozdíl mezi MFC a cepstrem je ten, že MFC má rovnoměrně rozloženou mel stupnici, což se blíží odezvě lidského sluchového systému víc, než lineárně rozložená frekvenční pásma, která jsou využívána v normálním spektru.

MFCC se vypočítá pomocí následujících kroků.

1. Nejdříve se aplikuje Fourierova transformace na signál.
2. Poté se převede výsledné spektrum pomocí logaritmu na Log-amplitudové spektrum.
3. Následně se použije na toto spektrum banka Mel filtrů (obr. 1.3 převzato z [6]).
4. Na závěr se aplikuje diskrétní kosinova transformace.

[1] grafy převzaty z [4]

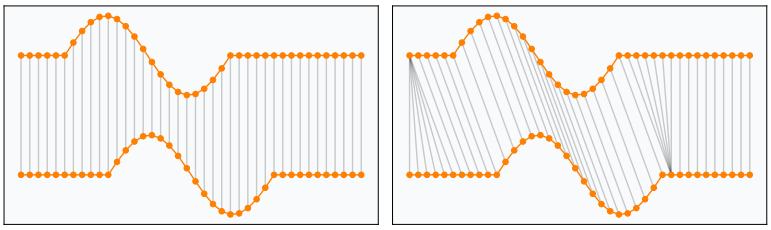


Obr. 1.3: Příklad banky Mel filtrů

1.2.3 Dynamická časová deformace

Dynamická časová deformace (Dynamic time warping – DTW) je algoritmus, který dokáže změřit podobnosti dvou časových řetězců, které se převážně liší pouze jejich rychlostmi.

Kdybychom při porovnávání dvou signálu použili klasickou Euklidovskou metriku, narazili bychom na problém, že časové rozdíly mezi signály nejsou shodné. DTW se tento problém snaží řešit tak, že hledá takové časové zarovnání, které minimalizuje Euklidovskou vzdálenost mezi zarovnanými sériemi (viz obr. 1.4 převzato z [7]). Také DTW využívá dynamického programování, to jest metoda pro efektivní řešení optimalizačních úloh. Rekurzivně rozdělí úlohu na menší části, které se později postupně řeší v nejoptimálnějším pořadí. Výsledky se používají jako vstupy do následujících úloh.



(a) (b)

Obr. 1.4: (a) Euklidovská metrika, (b) Dynamická časová deformace

V audio oboru se DTW využívá k synchronizaci audio stopy a na rozpoznání dvou obdobně znějících nahrávek, proto se ve své době používala na rozpoznání řeči. Tato metoda se poté postupně přestávala používat až do příchodu neuronových sítí a strojového učení, které vlilo této metodě „druhý dech“.

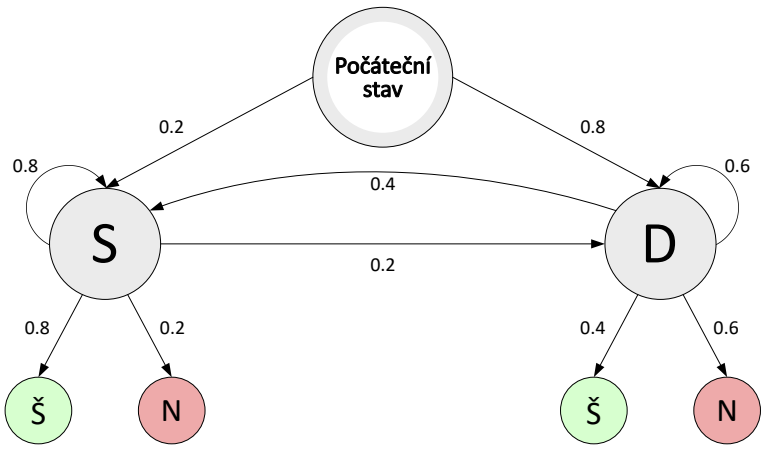
1.2.4 Skryté Markovovy modely

Markovovy modely jsou statistické modely, jejichž výstupem je posloupnost veličin nebo znaků. Tento model se snaží předpovědět následující stav procesu ze současného stavu. Tvrdí, že události před současným stavem jsou nepotřebné. Rozdíl mezi Markovovym modelem a skrytým Markovovým modelem (HMMs) je ten, že u Markovova modelu je stav pozorovatele a jeho výstup viditelný. U HMMs je stav pozorovatele skrytý, tím pádem každý stav má pravděpodobnostní vliv na výstup systému.

Skryté Markovovy modely mají hromadu využití nejen v audio oborech. Za zmínku patří například:

- finance,
  - předpovídání cen na burze,
- kryptoanalýza,
- biomedicína,
  - objevování motivů DNA,
  - predikce genů.

Nejznámější vysvětlení Markovova modelu spočívá v tom, že máme dva výchozí stavy: Slunečno (**S**) a Deštivo (**D**). Když je slunečno, je vyšší šance, že člověk bude šťasten (**Š**), a naopak, když je deštivo tak je vyšší šance, že člověk bude nevrlý (**N**). Z obr. 1.5 je vidno, že z počátečního stavu máme 80 % ku 20 % šanci, že bude deštivo. Když bude deštivo, tak máme 60 % ku 40 % šanci, že člověk bude nevrlý, a když bude slunečno tak máme 80 % ku 20 %, že člověk bude šťasten. Zároveň když bude jeden den slunečno, šance, že následující den bude opět slunečno, je 80 % ku 20 %, a když bude deštivo, šance jsou 60 % ku 40 %, že následující den bude opět deštivo. Tyto procentní hodnoty se zjistily pomocí pozorování posloupnosti různých stavů.



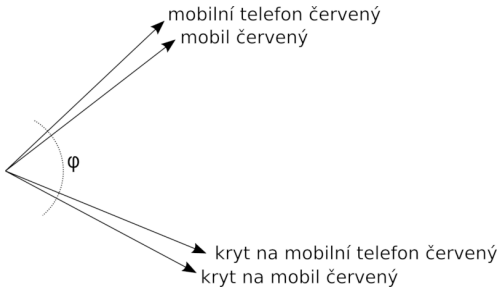
Obr. 1.5: Základní princip Markovova modelu

U rozpoznání řeči HMMs pracuje následovně.

1. Nejdříve se HMMs bude snažit přijít na to, jakým fonémem začíná věta. Jako příklad uvedeme větu: „Skákal pes“. HMMs se nejdříve podívá na první foném prvního slova. Tedy v našem případě */s/*. HMMs neví, že tento foném je */s/* a na základě pravděpodobnosti si tipuje, o jaký foném se jedná. Má na výběr z následujících možností: */s, z, t͡s, d͡z/*, jelikož znějí obdobně jako písmeno „s“. Podle možnosti s největší pravděpodobností se HMMs rozhodne, že naše první slovo začíná na písmeno „s“.
2. Následně HMMs zopakuje krok jedna, ale tentokrát pro druhý foném tohoto slova. Zjistí, že s největší pravděpodobností se jedná o písmeno „k“. Poté se podívá na to, jak pravděpodobný je stav, kdy po písmenu „s“ se nachází písmeno „k“. Zjistí, že je velice možné, že po „s“ následuje „k“. Poté se přesune na další foném a zjistí, že se jedná o písmeno „á“. Opět se podívá, jak moc pravděpodobné je to, že po písmenech „sk“ následuje písmeno „á“, a tak dále.
3. Při celém bodu 2. se HMMs snaží uhodnout, jaké slovo bylo vysloveno. V našem případě by pravděpodobně brzo poznal, že první slovo je slovo „Skákal“ a potom by uhodl, že druhé slovo je slovo „pes“. Ovšem může nastat stav, kdy si HMMs bude myslet, že se jedná o jiné slovo, než bylo vyřčeno. Opraví se tak, že následující slovo nebude dávat smysl, a z toho důvodu se vrátí zpátky k prvnímu slovu a bude v něm pokračovat. Názorný příklad tohoto problému je třeba slovo „Hladový“, kdy HMMs si ze začátku bude myslet, že se jedná a dvě slova: „Hlad“ a „ový“. Poté je ovšem spojí dohromady.

1.2.5 Jazykové modely

Jazykové modely pracují tak, že se snaží pochopit význam jednotlivých slov a díky tomu pochopit kontext celkového slovního spojení, ba dokonce i celých vět.



Obr. 1.6: Reprezentace jazykového modelu ve vektoru

Nejčastěji se k jazykovému modelu využívají neuronové sítě, které využívají vektor jako indikátor podobnosti jednotlivých slov. Počítač nedokáže pochopit, co slovní

spojení znamenají, ale dokáže rozlišit významově blízké texty a ty umístí ve vektorovém zápisu blíže k sobě. Naopak významově rozdílné texty posune dál od sebe [8].

Na obrázku 1.6 [2] můžeme vidět příklad výstupu jazykového modelu. Všimněme si, že slova podobného významu jsou blízko u sebe. I když „kryt na mobilní telefon červený“ a „mobilní telefon červený“ mají spoustu stejných slov, tak slovo „kryt“ mění kompletní význam celkového textu, a proto jsou tak vzdálené od sebe. Na určení vzdálenosti se používá úhel mezi vektory.

Nejznámější jazykové modely jsou [8]:

- Word2vec,
- fastText,
- BERT.

1.3 Strojové učení a neuronové sítě

1.3.1 Strojové učení

Strojové učení (Machine learning – ML) se snaží o to, aby se algoritmus dokázal sám zlepšovat a zdokonalovat na základě zkušeností, které pobral během procesu trénování. Je úzce spojováno se statistikou a výstup algoritmu je velice závislý na kvalitě vstupních dat. ML pracuje na hodně jednoduchém principu. V podstatě se dívá na věci, které dobře fungovali v minulosti, a říká si, že v budoucnosti tyto věci budou fungovat stejně dobře. Například: „Jelikož x % čeledí má geograficky oddělené druhy s různými barevnými variantami, tak zde existuje y% šance, že někde existují neobjevené černé labutě“ [9].

ML se nejčastěji řadí do 3 kategorií (obr. 1.7).

- Učení s učitelem
- Učení bez učitele
- Zpětnovazební učení

Učení s učitelem (Supervised learning – SL) funguje tak, že algoritmu jsou dodána vstupní data s příslušnými výstupními daty, které se po algoritmu budou očekávat [10]. Na základě znalosti výstupních dat si algoritmus upravuje vnitřní funkce tak, že přiřadí jednotlivým částem určitou váhu (jak moc daný parametr je důležitý k dosažení výsledku). Úspěšnost toho, jak moc efektivně odhaduje správné výsledky, se měří pomocí ztrátové funkce, kterou se snaží minimalizovat. Nejčastěji se algoritmy SL řadí do následujících typů:

- aktivní učení,
- klasifikace,
- regrese.

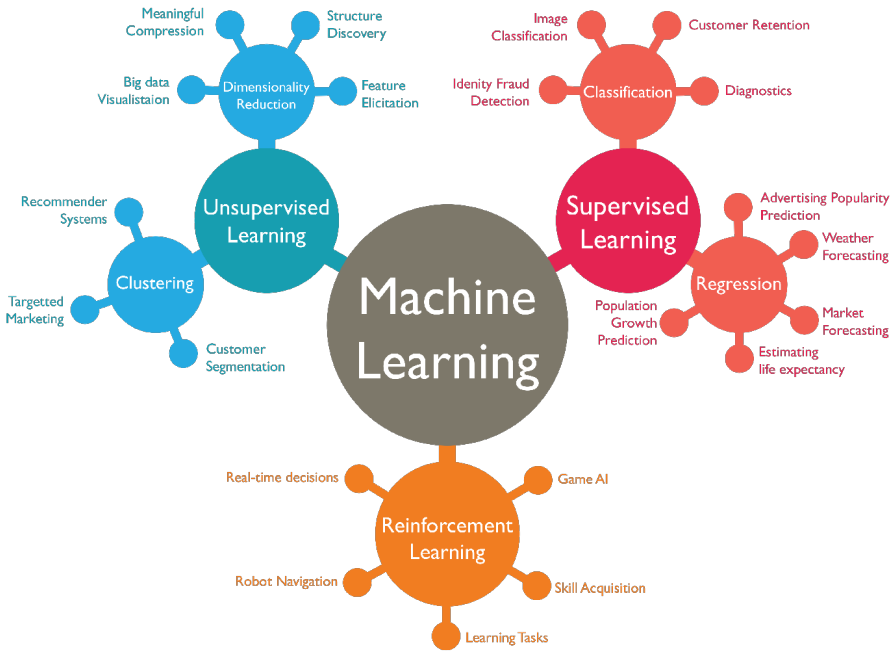
[2] převzato z [8]

Učení bez učitele (Unsupervised learning) je typ ML, který vůbec nevyužívá výstupní data. Algoritmu jsou poskytnuta pouze data vstupní, která nebyla nijak označena či kategorizována [10]. Jelikož učení bez učitele není závislé na zpětné vazbě (jako učení s učitelem), algoritmus se snaží přijít na spojitosti mezi vstupními daty. Nejznámější příklad tohoto učení je shluková analýza (clustering), která se snaží vstupní data roztřídit tak, aby byla v rámci stejného shluku podobná podle takových kritérií, která si ML samo vyhodnotí.

U **zpětnovazebního učení** (Reinforcement learning – RL) nejsou potřebná žádná vstupní data, jelikož oproti ostatním pracuje na úplně jiném principu. Tento algoritmus je v podstatě „vhozen“ do dynamického prostředí, ve kterém se snaží splnit zadaný úkol [11]. Podle úspěšnosti během procesu plnění zadaného cíle dostává zpětnou vazbu, na jejímž základě se snaží opět maximalizovat svoji úspěšnost. RL se používá převážně v teorii her a jeho prostředí je nejvíce reprezentováno ve formě Markovových rozhodovacích procesů (Markov decision process – MDP), které úzce souvisí s dynamickým programováním.

Také se ještě uvádí učení s částečným učitelem, což je kombinace SL a učení bez učitele. V některých případech je prokázané, že když malá část vstupních dat nejsou kategorizovaná, může to vést ke značnému zlepšení přesnosti učení.

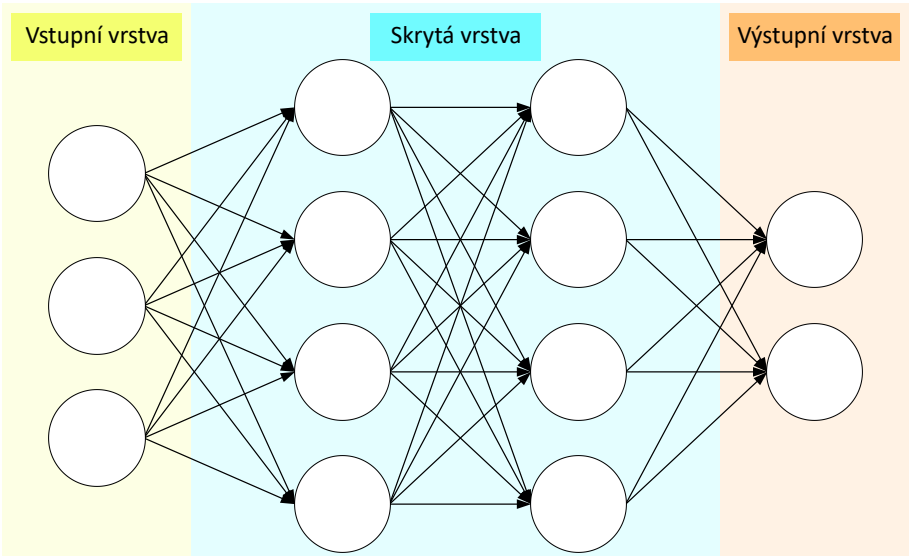
Strojové učení je často zaměňováno s pojmem umělá inteligence (Artificial intelligence – AI), jelikož v zásadě dělají jednu a tu samou věc – učí se na základě určitého pozorování. Rozdíl mezi nimi je ten, že zatímco AI se aktivně učí a podniká akce aby maximalizoval svoji úspěšnost, ML se nečinně učí a pozoruje.



Obr. 1.7: Větvení strojového učení, převzato z [12]

1.3.2 Neuronové sítě

Neuronové sítě (Neural networks – NN), nebo také umělé neuronové sítě (Artificial neural networks – ANN), jsou systémy, jenž fungují na principu, který se nejvíce blíží tomu, jak funguje a operuje náš mozek [13]. Spadá pod strojové učení a také se rozděluje na učení s a bez učitele. ANN má v sobě skrytou vrstvu uzlů (neuronů), které se, stejně jako synapse v mozku, propojí s ostatními souvisejícími uzly. Tato spojení se nazývají hrany (edges). Každý z těchto neuronů v sobě ukrývá určitou nelineární funkci, jenž se vypočítá ze svých vstupních dat a jejichž výstup je opět vstupem pro další neuron. Každý z těchto neuronů a hran má svoji váhu, která se upravuje v průběhu učení, kdy algoritmus rozhoduje, jak moc je daná složka potřebná. Neurony jsou sdružovány do vrstev. Může jich být i několik za sebou. Obvykle první vrstva bývá vstupní a výstupní vrstva, bývá nejčastěji poslední (obr. 1.8).



Obr. 1.8: Příklad neuronové sítě

Existuje několik druhů neuronových sítí. Za zmínění stojí:

- perceptron (P),
- vícevrstvý perceptron (MP),
- neuronová síť s dopřednou vazbou (FF),
- neuronová síť s radiální bází (RBF),
- modulární neuronová síť (MNN),
- rekurentní neuronová síť (RNN),
- konvoluční neuronová síť (CNN).

Perceptron je jeden z nejjednodušších a nejstarších modelů neuronu. Je to nejmenší jednotka neuronové sítě. Řadí se do SL, a jelikož třídí data pouze do dvou kategorií, jedná se o binární klasifikátor. Perceptron může být implementovaný jako funkce logických hradel (AND, OR, XOR, atd.).

Vícevrstvý perceptron je složený z vrstev několika perceptronů. Každý uzel je propojený se všemi uzly v další vrstvě a vstupní i výstupní vrstvy jsou napojeny na několik skrytých vrstev (více než 1). Jelikož je jeho směr šíření oboustranný, je to dobrá volba pro aplikaci hlubokého učení (deep learning).

Neuronová síť s dopřednou vazbou je nejjednodušší forma neuronových sítí, jelikož směr šíření je pouze jednosměrný. Tato forma nemá zpětnou vazbu, a proto jsou zde váhy statické. Skládá se ze vstupní, skryté a výstupní vrstvy. Skrytá vrstva není podmínkou a místo ní se zde můžou nacházet další vstupní či výstupní vrstvy. Na základě toho se rozděluje na jednovrstvou a vícevrstvou.

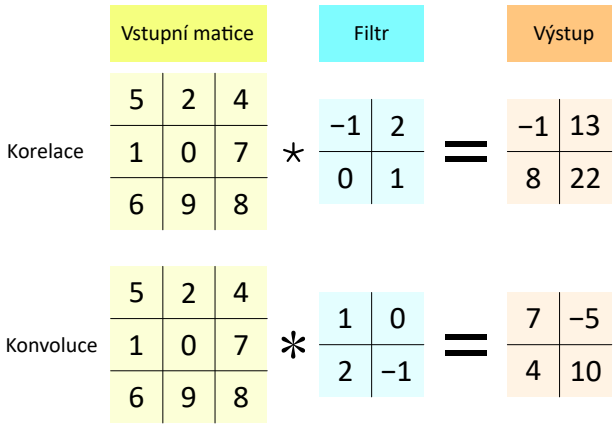
Neuronová síť s radiální bází se skládá ze vstupního vektoru, za nimž se nachází vrstva neuronů RBF, a končí výstupní vrstvou s jedním uzlem pro každou kategorii. Každý neuron RBF porovnává vstupní vektor se svým prototypem a vypisuje hodnotu, jak moc je tento prototyp podobný vůči vstupnímu vektoru. Výstupní vrstva je vytvořena z neuronů.

Modulární neuronová síť má několik různých sítí, které fungují nezávisle na sobě. Každá tato síť vykoná nějakou dílčí část celkového úkolu. Sítě mezi sebou vůbec neinteragují a ani mezi sebou nemají zpětnou vazbu. Výhoda těchto sítí je rychlost při komplexnějších výpočtech, jelikož se rozdělí na nezávislé komponenty a poté se vypočítávají v jiných vnitřních sítích.

Rekurentní neuronová síť je jedna ze dvou nejčastějších NN. RNN ukládá výstup vrstvy, který následně vrací zpátky na vstup, a pomáhá tak předpovídat výsledky. Nejčastěji se skládá z první vrstvy, neuronové sítě s dopřednou vazbou, po které následuje vrstva rekurentní neuronové sítě. V této RNN vrstvě si NN některé informace pamatuje z předchozího cyklu. Pomocí toho dokáže RNN mírně upravovat svůj proces učení v případě, pokud výsledek vyjde jinak, než podle očekávání.

Konvoluční neuronová síť je nejčastější typ neuronových sítí v oblastech vizuální analýzy. Jako jediná NN využívá trojrozměrné uspořádání neuronů namísto dvojrozměrného. CNN je v podstatě upravovaná verze vícevrstvého perceptronu. Dokáže upravovat svoje parametry během trénování, jako například *weight decay, dropout, skipped connection, apod.*. CNN obsahuje konvoluční vrstvu. V této vrstvě jsou neurony, které zpracovávají výsledek pouze z malé části. Síť si rozkouskuje obraz na několik pravidelných částí, které poté projdou přes filtry (kernel), a takto proces postupuje dál, dokud nedojde k úplnému zpracování. Konvoluce se často zaměňuje s korelací. Jejich principy jsou velice podobné, ovšem u konvoluce se otočí matice filtrů o 180° (obr. 1.9). Korelace je označována symbolem pěticípé hvězdy a

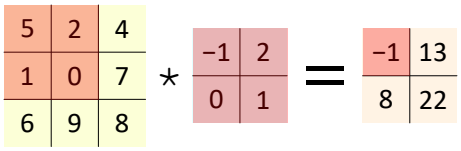
konvoluce symbolem hvězdy se šesti cípy. Celý název korelace se uvádí jako křížová korelace (cross correlation).



Obr. 1.9: Rozdíl mezi korelací a konvolucí

Korelace se rozděluje do dvou druhů:

1. Platná korelace (valid correlation) – Počítá se tak, že se filtr použije pouze na celá vstupní data a nepřesahuje hranice těchto dat. Výstup se počítá následovně:

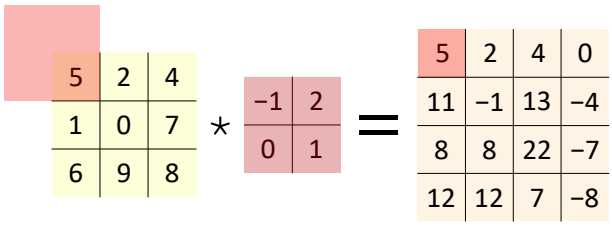


$$5 \cdot -1 + 2 \cdot 2 + 1 \cdot 0 + 0 \cdot 1 = -1.$$

Podle stejného principu se dopočítají zbylé výsledky výstupu:

$$2 \cdot -1 + 4 \cdot 2 + 0 \cdot 0 + 7 \cdot 1 = 13,$$
$$1 \cdot -1 + 0 \cdot 2 + 6 \cdot 0 + 9 \cdot 1 = 8,$$
$$0 \cdot -1 + 7 \cdot 2 + 9 \cdot 0 + 8 \cdot 1 = 24.$$

2. Plná korelace (full correlation) – v základě se počítá stejně jako platná korelace, s tím rozdílem, že se filtr aplikuje na celá data, nezávisle na tom, že přesáhne hranice vstupních dat. Výsledkem je rozměrově matice větší, než byla ta vstupní.



Plná korelace se vypočítá následovně:

$$
\begin{aligned}
5 \cdot 1 &= 5, \\
5 \cdot 0 + 2 \cdot 1 &= 2, \\
2 \cdot 0 + 4 \cdot 1 &= 4, \\
4 \cdot 0 &= 0, \\
5 \cdot 2 + 1 \cdot 1 &= 11, \\
5 \cdot -1 + 2 \cdot 2 + 1 \cdot 0 + 1 \cdot 1 &= -1, \\
2 \cdot -1 + 4 \cdot 2 + 0 \cdot 0 + 7 \cdot 1 &= 13, \\
4 \cdot -1 + 7 \cdot 0 &= -4, \\
1 \cdot 2 + 6 \cdot 1 &= 8, \\
1 \cdot -1 + 0 \cdot 2 + 6 \cdot 0 + 9 \cdot 1 &= 8, \\
0 \cdot -1 + 7 \cdot 2 + 9 \cdot 0 + 8 \cdot 1 &= 24, \\
7 \cdot -1 + 8 \cdot 0 &= -7, \\
6 \cdot 2 &= 12, \\
6 \cdot -1 + 9 \cdot 2 &= 12, \\
9 \cdot -1 + 8 \cdot 2 &= 7, \\
8 \cdot -1 &= 8.
\end{aligned}
$$

Výsledná matice konvoluční vrstvy se počítá tak, že se použije platná korelace vstupních dat a filtrů a k výsledné hodnotě se přičte zkreslení (biases). Jelikož vstupní matice je trojrozměrná, filtry musí být taktéž trojrozměrné. Vrstva může mít několik filtrů, ale všechny musejí mít stejnou hloubku jako vstupní matice. Ke každému filtru se přiřadí dvojrozměrné zkreslení, které bude mít stejný rozměr jako výsledná matice. Hloubka výsledné matice určuje počet filtrů. Pokud tedy bude použitý pouze jeden filtr, výsledná matice bude dvojrozměrná.

2 Zpracování řeči

2.1 Detekce hlasové aktivity

Detekce hlasové aktivity (Voice activity detection – VAD) je systém pro zjištění přítomnosti či nepřítomnosti lidské řeči v audio signálu [14].

Hlavní využití VAD je při rozpoznání řeči, pomocí této detekce, jsme schopni rozdělit audio signál na část, v němž je obsažena lidská řeč a na část kde je „ticho“. Tato funkce se využívá v několika odvětvích, převážně pro ušetření výpočetního výkonu. Využívá se například v aplikacích VoIP (Voice over Internet Protocol), kde zabraňuje poslání paketů ticha, nebo při již zmíněném rozpoznání řeči, kdy dokáže vyřadit rámce, které neobsahují přítomnost lidské řeči, pro odlehčení výkonu následujícího zpracování.

Algoritmů pro VAD je několik druhů. Všechny balancují nad kompromisu mezi délkou latence, citlivostí, přesností a výpočetní náročností. Některé algoritmy poskytují více analýz, například zda-li je řeč hlasitá či potichu. Algoritmy VAD také nebývají závislé na jazyku.

Mezi nejčastější přístupy k detekci hlasové aktivity patří:

- Energetické prahování (Energy thresholding) – řečový signál není stacionární, tedy sekce signálu obsahující řeč bude s největší pravděpodobností mít vyšší energetickou hodnotu než sekce, která řeč neobsahuje. Na základě této myšlenky nastavíme vhodnou energetickou hranici. Jakmile určitý úsek překročí tuto energetickou hranici, můžeme předpokládat, že tento úsek obsahuje mluvené slovo.
- Další z přístupů je signál analyzovat několika různými způsoby, které určí vlastnosti. Na základě těchto vlastností VAD rozhodne, zda zadaný signál obsahuje řeč. Mezi tyto způsoby patří třeba již zmíněné energetické prahování, lineární predikce, výraznější základní frekvence v rozsahu 80 Hz až 450 Hz nebo také využití MFCC pro další určení potřebných vlastností. Na závěr je potřeba určit, nejčastěji pomocí rozhodovacího stromu, které z těchto vlastností musí splňovat signál, aby platilo, že obsahuje řečovou část.
- Strojové učení – pomocí strojového učení lze natrénovat systém pro rozpoznání přítomnosti mluvy v nahrávce. U těchto systémů je ovšem zapotřebí si uvědomit, že účelem VAD je především snížit výpočetní náročnost přístroje, proto by měl mít systém nízkou složitost, jednodušší návrh a menší počet parametrů. Toto rozhodnutí do jisté míry ovlivní kvalitu výsledku, ale to jest ten kompromis, který musíme akceptovat.

2.2 Odstranění šumu

Odstranění šumu (Noise reduction) je proces, který se snaží odstranit nežádoucí složku (obvykle různý typ šumu) ze signálu. Algoritmy redukce šumu můžou způsobit mírné zkreslení původního signálu a jsou rozdělovány podle typu aplikace na *audio noise reduction* a *image noice reduction*.

Pro **odstranění šumu pro obrazové signály** se nejčastěji využívá konkrétní typ a forma filtrů. Například pomocí konvoluce se spojí původní obraz s obrazem, na němž byl aplikován filtr dolní pásmové propusti, nebo se může na celý signál aplikovat mediánový filtr. V poslední době se také začala využívat pro odstranění šumu variace strojového učení, a to jak z obrazového signálu, tak z audio signálu.

Odstranění šumu pro audio signál se dosahuje několika způsoby. Nejjednodušší forma redukce je použití lineárních či nelineárních filtrů, které pracují v časově-frekvenční oblasti a které jsou také nazývány jako časově-frekvenční filtry [15]. Nejpoužívanější algoritmy v programech zabývajících se audio signály bývají ty, jenž využívají buď šumové brány (Noise gate), nebo šumového omezovače (Noise limiter). Spousta programů ovšem obsahuje více než jednu metodu redukce šumu.

Noise gate funguje na principu výpočtu spektogramu signálu a následném odhadu prahu šumu pro každé frekvenční pásmo tohoto signálu. Tento práh se poté využije k výpočtu masky, která oddělí šum nacházející se pod touto hranicí prahu. Rozděluje se na následující metody:

- stacionární redukce šumu – práh šumu je stejný po celou dobu nahrávky,
- nestacionární redukce šumu – průběžně se aktualizuje hodnota prahu šumu po celou dobu nahrávky.

Noise limiter se kromě programů v elektronické formě využíval také v analogové formě, například při zvukových systémech v autě, popřípadě v systémech pro nahrávání mikrofonů. Pracuje na principu porovnání jednoho snímku s druhým snímkem signálu a odstraňuje drobné projevy, které nejsou ve snímcích stejné.

Je zapotřebí zmínit častokrát zaměňovanou techniku redukce šumu, a to aktivní potlačení šumu/hluku (Active noise cancellation). Tato metoda je využívána především ve sluchátkových systémech, ve kterých se potlačení hluku provádí tak, že mikrofony zaznamenávají zvuky vně i uvnitř sluchátek a následně jedné z těchto stop otočí fázi. Tímto dojde k neutralizaci hlukové složky. Tato metoda ovšem není považována za noise reduction metodu z důvodu fundamentálně rozdílné funkčnosti.

3 Návrh systému pro rozpoznání hlasových povelů

Systémy pro rozpoznání hlasových povelů se nejčastěji skládají z následujících bodů.

1. Vybrání nahrávky, která má být klasifikována.
2. Úprava této nahrávky do tvaru vhodný pro klasifikaci.
3. Samotné rozpoznání (klasifikace).

Každý systém pro rozpoznání hlasových povelů, které nejsou zaměřené na kontinuální mluvu, bude obsahovat v nějaké formě tyto kroky. V případě, že se jedná o implementaci v reálném čase, může být první krok zaměněn o výsledek řečového signálu, který byl rozpoznaný pomocí VAD.

3.1 Návrh systému pomocí neuronové sítě

Z metod uvedených v teoretické části této práce bylo po domluvě s vedoucím práce rozhodnuto, že metoda, která bude využita k vytvoření systému pro rozpoznání hlasových povelů, bude metoda neuronových sítí. V této práci byly použity konkrétně konvoluční neuronové sítě v programovacím jazyce python.

Kompletní proces, použitý v této práci, pro vytvoření neuronové sítě, která dokáže rozpoznávat hlasové povely se dá rozdělit do následujících kroků:

1. obstarání audio nahrávek pro trénování neuronové sítě,
2. úprava nahrávek do jednotné formy a následné převedení do MFCC,
3. vytvoření vhodného datasetu,
4. vytvoření přijatelné architektury neuronové sítě,
5. a výsledné trénování neuronové sítě.

Toto ovšem zahrnuje pouze tvorbu samotného klasifikátoru (Bod 3. v předchozí sekci). Celkový systém lze najít na blokovém schématu (obr. 3.1). Tento systém se skládá z již zmíněného VAD, který zvýrazní úseky, ve kterých si myslí, že se nachází řečový signál. Poté se pomocí filtru (na způsob mediánového filtru) tento signál vyčistí o úseky, které pravděpodobně neobsahují řečový signál a byly zaměněné s úseky, které tento signál obsahují. Následně se tato nahrávka vystřihne z původního signálu. V tomto kroku je také nahrávka upravena, aby později měla vhodný tvar pro neuronovou síť. V neposlední řadě se nahrávka převede na MFCC, jelikož neuronová

Obr. 3.1: Blokový diagram systému pro rozpoznání hlasových povelů pomocí NN

síť by z klasického signálu nedokázala rozeznat vyslovená slova mezi sebou. Pravděpodobně by dokázala zjistit, jestli se v nahrávce nachází řeč či nikoli, ale tento problém je již vyřešený algoritmem VAD. Tyto koeficienty jsou nakonec vyslány do klasifikátoru, tedy neuronové sítě, kde síť pomocí výsledných příznaků zjistí, které slovo se nachází v nahrávce. Výsledkem neuronové sítě je pole hodnot. Index nejvyšší hodnoty v tomto poli odkazuje na index slova, u kterého neuronová síť odhaduje, jestli se nachází v dané nahrávce.

Například pokud výsledný index vypadá následovně:

$$vysledny\_index = [0, \quad 0.12, \quad 0.15, \quad 0.63, \quad 0, \quad 0.02, \quad 0.05, \quad 0.03, \quad 0, \quad 0],$$

neuronová síť si na 63 % myslí, že slovo, které se nachází na nahrávce přivedené do této sítě, je slovo *Off*. Zároveň zde je možné vidět na kolik procent slovo v nahrávce obsahuje jiné slovo. Například NN si myslí na 15 %, že se jedná o slovo *On*, nebo na 12 %, že se jedná o slovo *Left*.

Jednotlivá slova a jejich příslušné indexy lze najít v tabulce 3.1.

Tab. 3.1: Tabulka povelů a jejich příslušných indexů

Povel	Index
Down	0
Left	1
No	2
Off	3
On	4
Right	5
Stop	6
Up	7
Yes	8
Zero	9

4 Návrh architektury neuronové sítě

4.1 Vstupní data

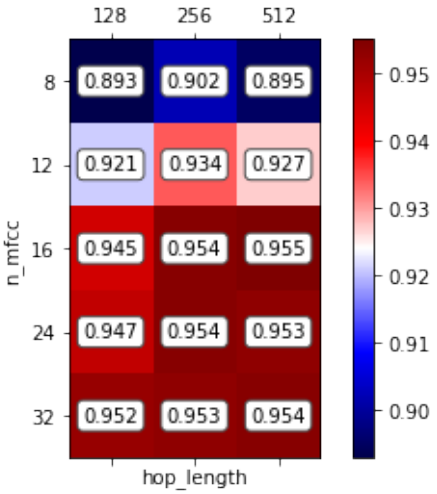
Pro trénování neuronové sítě je potřeba obrovské množství trénovacích dat, aby NN fungovala co nejlépe. Je také důležité, aby tato data byla nejen kvalitní, ale také co nejvíce rozdílná, aby NN neměla problém rozeznat i obtížné a rozdílnější příklady. Jelikož je cílem této bakalářské práce rozpoznání deseti slov, muselo by se ručně nahrát tisíce nahrávek různě znějících stejných deseti slov, abychom měli dostatečné množství dat na trénování NN. Z tohoto důvodu byla využita databáze slov `speech_commands_v0.01.tar.gz` od společnosti Google. Tato databáze obsahuje databázi 65 000 jednosekundových nahrávek 30 různých slov. Z těchto nahrávek byla využita následující slova: *down* (2 359 nahrávek), *left* (2 353 nahrávek), *no* (2 375 nahrávek), *off* (2 357 nahrávek), *on* (2 367 nahrávek), *right* (2 367 nahrávek), *stop* (2 380 nahrávek), *up* (2 375 nahrávek), *yes* (2 377 nahrávek) a *zero* (2 376 nahrávek). Všechny nahrávky jsou namluvené různými osobami většinou dospělého věku a u některých se nachází mírný šum v pozadí.

4.2 Převedení na MFCC

Jelikož neuronová síť nedokáže „pochopit“ samotné zvukové nahrávky, je potřeba je převést do formy, kdy je schopna s nimi začít pracovat. Ne všechny nahrávky mají přesně jednu sekundu, a proto je potřeba tyto nahrávky mírně upravit. Všech 23 686 nahrávek je tedy zapotřebí unifikovat.

K převedení nahrávek na MFCC byla využita knihovna `librosa`, která obsahuje funkci `mfcc`. Po několika testování byl počet mel koeficientů nastavený na nejoptimálnější hodnotu, a to na `n_mfcc = 16`. Délka skoku byla nastavena na `hop_length = 512`. Tyto hodnoty byly kompromisem mezi množstvím MFCC vzorků a výslednou přesností trénování. Na obrázku 4.1 je patrné, že od nastavených hodnot se přesnost o moc nemění, a zároveň námi zvolené hodnoty mají nejmenší rozměry výsledné matice, a to `[44, 16]`.

K tomu, aby měly všechny výsledné matice z nahrávek stejné rozměry, bylo potřeba upravit je ještě předtím, než se na ně aplikuje funkce `mfcc`. Nahrávky, které byly delší než jedna sekunda, byly pomocí funkce `time_stretch` zúženy, a nahrávky, které byly kratší než jedna sekunda, byly naopak doplněny o nuly.



Obr. 4.1: Vliv hodnot `n_mfcc` a `hop_length` na `val_accuracy`

```
time_skretch_rate = len(signal)/sample_rate
if time_skretch_rate > 1:
    signal = librosa.effects.time_stretch(signal,
                                          time_skretch_rate)
elif time_skretch_rate < 1:
    array = np.full(sample_rate-len(signal),
                                          np.float32(0))
    signal = np.concatenate([signal,array])
```

Funkce `load` z knihovny `librosa` zajišťovala převedení nahrávek do pole hodnot, se kterými se dalo následně pracovat. Tato funkce je ovšem pomalejší oproti ostatním funkcím, které plní stejný účel (například funkce `wavfile.read` z knihovny `scipy.io`). Jeden z důvodů je fakt, že funkce `load` automaticky převzorkovává nahrávky na námi zvolenou vzorkovací frekvenci (základně 22 050 Hz). To ovšem není potřeba, a tak by bylo rozumnější zvolit druhou rychlejší funkci. Důvod, proč i přesto byla použita funkce `load`, je ten, že tato funkce výsledné pole hodnot vrací v hodnotách s pohyblivou řádkovou čárkou (floating-point number). Funkce `mfcc` vyžaduje, aby vstupní signál byl v tomto tvaru čísel. Funkce `wavfile.read` ukládá hodnoty v celočíselném tvaru, a proto jej nelze využít pro tuto práci.

Na závěr tohoto procesu se všechny výsledné matice přiřadily do slovníku pod příslušné klíčové jméno. Klíčovými jmény jsou výrazy obsahující zvolená slova nahrávek. Finální slovník byl poté uložen do `json` souboru pro budoucí práci a možnou manipulaci.

4.3 Tvorba datasetu

Nyní je potřeba zpracovat předešlá data tak, aby byla vytvořena vhodná databáze pro trénování.

Jelikož byla tato data ukládána ze slovníku, kde jsou klíče jednotlivých slov napsány ve formě řetězce, potřebujeme tato klíčová slova vhodně oindexovat. Je to z toho důvodu, že neuronová síť nedokáže operovat s řetězcem slov.

$$data = \begin{Bmatrix} \text{"Down"} & : & [\text{MFCC 1, MFCC 2, ..., MFCC } n] \\ \text{"Left"} & : & [\text{MFCC 1, MFCC 2, ..., MFCC } n] \\ \text{"No"} & : & [\text{MFCC 1, MFCC 2, ..., MFCC } n] \\ \text{"Off"} & : & [\text{MFCC 1, MFCC 2, ..., MFCC } n] \\ \text{"On"} & : & [\text{MFCC 1, MFCC 2, ..., MFCC } n] \\ \text{"Right"} & : & [\text{MFCC 1, MFCC 2, ..., MFCC } n] \\ \text{"Stop"} & : & [\text{MFCC 1, MFCC 2, ..., MFCC } n] \\ \text{"Up"} & : & [\text{MFCC 1, MFCC 2, ..., MFCC } n] \\ \text{"Yes"} & : & [\text{MFCC 1, MFCC 2, ..., MFCC } n] \\ \text{"Zero"} & : & [\text{MFCC 1, MFCC 2, ..., MFCC } n] \end{Bmatrix} \begin{matrix} index : 0 \\ index : 1 \\ index : 2 \\ index : 3 \\ index : 4 \\ index : 5 \\ index : 6 \\ index : 7 \\ index : 8 \\ index : 9 \end{matrix}$$

Poté bylo potřeba spojit MFCC nahrávek s příslušným indexovým číslem příslušného klíče. Hodnoty těchto indexů lze najít v tabulce 3.1, která se nachází v kapitole 3.1. Výsledný list s názvem `training_data` vypadal následovně:

$$training\_data = \begin{bmatrix} [\text{ MFCC nahrávky "Down 1", } 0], \\ [\text{ MFCC nahrávky "Down 2", } 0], \\ \dots \\ [\text{ MFCC nahrávky "Down } n\text{", } 0], \\ [\text{ MFCC nahrávky "Left 1", } 1], \\ [\text{ MFCC nahrávky "Left 2", } 1], \\ \dots \\ [\text{ MFCC nahrávky "Left } n\text{", } 1], \\ \dots \\ [\text{ MFCC nahrávky "Zero 1", } 9], \\ [\text{ MFCC nahrávky "Zero 2", } 9], \\ \dots \\ [\text{ MFCC nahrávky "Zero } n\text{", } 9] \end{bmatrix}.$$

Je nezbytné data nahodile zamíchat. Koná se tak z toho důvodu, že neuronová síť by se jinak trénovala z „extrému do extrému“. Nejdříve by se natrénovala na všech nahrávkách slova „Down“ a až poté by přešla na nahrávky slova, které by bylo další v pořadí. Tento způsob trénování je neefektivní, a proto použitím funkce `shuffle` z knihovny `random` náhodně zamícháme toto pořadí. Výsledná data, jenž byla nahodile rozložená, zamezí velikým výkyvům při trénování NN.

Ještě bylo potřeba uložit zvlášť MFCC a indexy do jiných proměnných z důvodu vyžadovaných parametrů konvoluční neuronové sítě. Do proměnné `y` byly uloženy indexy a do proměnné `X` byly uloženy příslušná MFCC data, která byla následně pomocí knihovny `numpy` převedena na pole hodnot.

4.4 Návrh a ladění konvoluční neuronové sítě

Pro tvorbu konvoluční neuronové sítě bylo využito rozhraní pro programování aplikací (Application programming interface dále jako API) `Keras`, která spadá pod velice známou API TensorFlow. TensorFlow je nejpoužívanější API k budování systémů využívající strojového učení a neuronových sítí. Keras patří pod tuto API a zaměřuje se na jednoduché a velmi rychlé aplikování strojového učení, popřípadě neuronových sítí.

Neuronová síť požaduje pro trénování následující proměnné: `x_train, y_train, x_test, y_test`. Proměnné s indexem `train` jsou všechna data, na kterých se bude neuronová síť učit, a s indexem `test` jsou data, na kterých bude NN testovat, jestli předpovídá správně. Poměr mezi množstvím těchto dat byl 90 % ku 10 % pro trénovací data, tento poměr určuje proměnná `trainborder`.

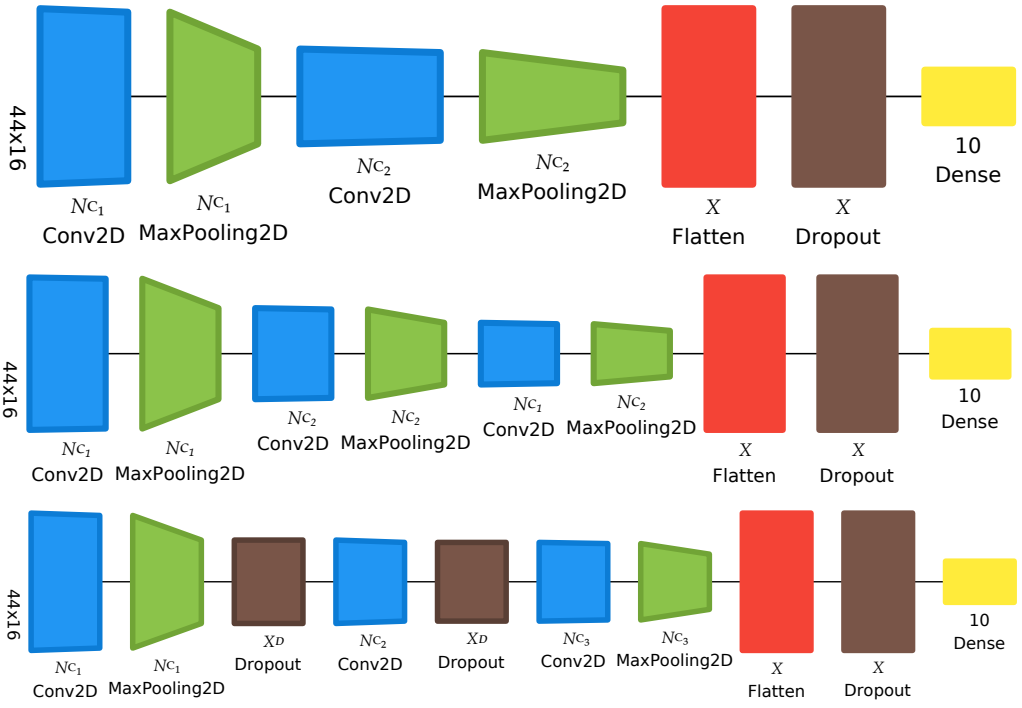
```
trainborder = int(len(x)*0.90)
x_train = X[0:trainborder]
y_train = y[0:trainborder]
x_test = X[trainborder:]
y_test = y[trainborder:]
```

Také bylo potřeba expandovat rozměr dat a zároveň převést data z jednotlivých vektorů na binární matice, aby je síť mohla zpracovat.

```
x_train= np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
y_train= keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

Oproti semestrální práci, kde byla využita předem připravená neuronová síť s názvem Simple MNIST convnet od tvůrce Keras API pod aliasem fchollet, bylo vytvořeno několik prototypů modelů neuronových sítí za účelem najít nejlépe pracující architekturu s nejvíce vhodnými hyperparametry.

K tomuto hledání byla využita knihovna `keras-tuner` (použito z [16]). Pomocí funkce `RandomSearch` z této knihovny bylo možné zautomatizovat proces hledání nejlepších hodnot parametrů a hyperparamentrů. Tato funkce umožňuje kontrolovat zvolený parametr (nejčastěji `val_accuracy`), a zároveň dokáže náhodně upravovat hodnotu jiných zvolených parametrů mezi cykly. U těchto parametrů se nastavovaly mezní hodnoty (spodní a horní hranice, které parametr neměl překročit) a hodnota, o kolik se parametr mohl zvětšovat či zmenšovat. Také bylo možné upravit datový typ, ale ten byl ponechán na celočíselných hodnotách, protože zvolené parametry nemohou operovat v jiných datových typech než celočíselných. Po uplynulých trénovacích epochách si funkce uloží všechny tyto proměnné hodnoty a přesune se na další probíhající cyklus, ve kterém se opět náhodně upraví hodnoty zvolených parametrů. Počet těchto cyklů je možné nastavit. Takto bylo natrénováno přes 300 unikátních neuronových sítí přibližně na 20prototypech modelů. Všechny tyto neuronové sítě se trénovaly na 20 epoch. Na obrázku 4.2 se nachází pár architektur těchto prototypů, které byly vizualizovány pomocí nástroje Net2Vis [17].

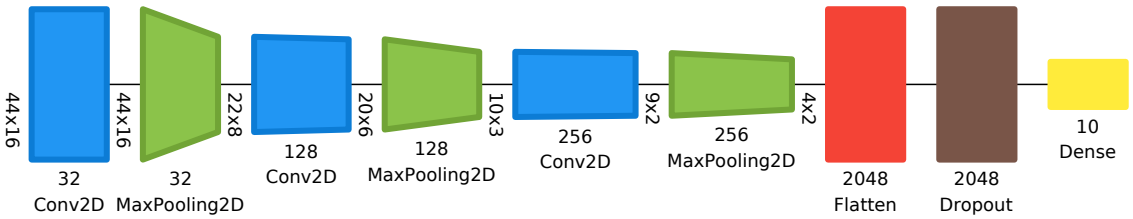


Obr. 4.2: Vizualizace architektury prototypů modelů NN

Po všech testech (přibližně 100 výpočetních hodin) byla nejpřesnější neuronová síť následující:

```
model = keras.Sequential(
[
 keras.Input(shape=input_shape),
layers.Conv2D(32, kernel_size=(4, 4), activation="relu",
                                          padding="same"),
layers.MaxPooling2D(pool_size=(2, 2)),
layers.Conv2D(128,kernel_size=(3, 3), activation="relu"),
layers.MaxPooling2D(pool_size=(2, 2)),
layers.Conv2D(256,kernel_size=(2, 2), activation="relu"),
layers.MaxPooling2D(pool_size=(2, 1)),
layers.Flatten(),
layers.Dropout(0.5),
layers.Dense(num_classes, activation="softmax"),
])
```

Na obrázku 4.3 se nachází vizualizace architektury této neuronové sítě. Po dvaceti epochách trénování neuronové sítě `val_accuracy` dosahovala hodnoty 0,948.



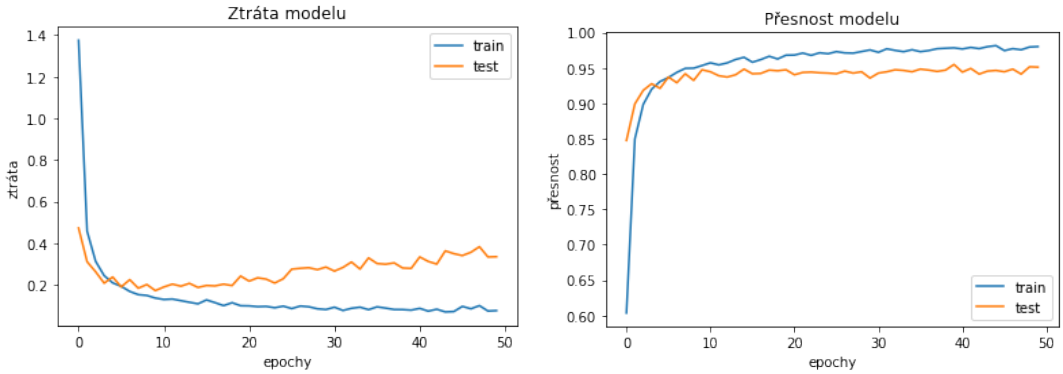
Obr. 4.3: Vizualizace architektury neuronové sítě

Finální model byl zkompilovaný s metrikou „accuracy“ a nastavením ztrátové funkce na „categorical_crossentropy“. Účelem ztrátových funkcí je vypočítat veličinu, kterou by se měl model snažit během trénování minimalizovat. Byl využit „Adam“ jako optimalizátor, který se řadí mezi stochastické metody gradientního sestupu. Tato metoda je založena na adaptivním odhadu momentů prvního a druhého řádu.

Výsledná síť se trénovala na 50 epoch s velikosti parametru `batch_size = 64`, který určuje kolik dat se pošle zároveň na trénování neuronové sítě. Validační rozdíl byl 0,1. Kdyby byla síť trénována na více epochách, razantně by se zvyšovala hodnota `val_loss`. Pomocí této hodnoty je možné určit, zda výsledný model je dobře přizpůsobeným modelem a nepatří do kategorie *Overfit* nebo *Underfit* modelu.

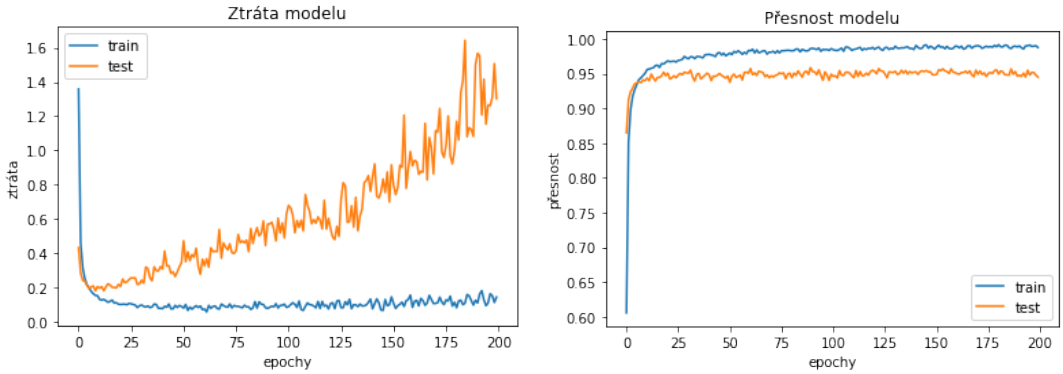
- Overfit model – je model, který se příliš moc vytrénuje na trénovací datové sadě. Vede si dobře na trénovacích datech, ale má nedostatečné výsledky v datech, které se nachází mimo tuto trénovací sadu.
- Underfit model – tento model je nejen špatně vytrénovaný na trénovací datové sadě, takže má problém rozpoznávat data z této sady, ale vede si i špatně na datech, které jsou i mimo tuto sadu.
- Good fit model – tento model se vhodně natrénoval z trénovací datové sady a zároveň dobře rozpoznává data mimo tuto sadu.

Ve finální neuronové síti *Test Loss* dosahuje hodnoty 0,318 a přesnost modelu *Test Accuracy* hodnoty 0,956. Tyto hodnoty byly zjištěny pomocí funkce `evaluate`. Na obrázku 4.4 je zobrazena změna těchto hodnot v závislosti na epochách.



Obr. 4.4: Grafy změň hodnot ztráty a přesnosti modelu v závislosti na 50epochách

Na obrázku 4.5 je zobrazeno, jak by model vypadal, kdyby se trénoval na 200 epochách. Lze vypozorovat, jak s každou následující epochou roste ztráta modelu, ale přesnost modelu se minimálně mění. Tento model by patřil do kategorie *Overfit model*.



Obr. 4.5: Grafy změň hodnot ztráty a přesnosti modelu v závislosti na 200epochách

5 Testování neuronové sítě

Výsledná neuronová síť (z předchozí sekce) byla trénovaná na datasetu, který se skládal převážně z čistě znějících nahrávek, tj. bez žádných ruchů, popřípadě z pár nahrávek ve kterých se objevoval mírný šum.

Otázkou tedy zůstává, jak by se sestavená neuronová síť chovala, kdyby se trénovací dataset skládal z nahrávek, které obsahují nějakou formu ruchu. Nahrávky ruchů byly použity ze stejné databáze jako nahrávky jednotlivých slov, tedy z databáze `speech_commands_v0.01.tar.gz` od společnosti Google. Tato databáze, kromě již zmíněných nahrávek slov, obsahovala 6 různých nahrávek ruchů. Mezi tyto ruchy patří: *white_noise*, *pink_noise*, *doing_the_dishes*, *exercise_bike*, *running_tap* a *dude_miaowing*. Poslední nahrávka z této databáze nebyla použita.

Pro vyhnutí manuálního přidávání ruchů do nahrávek byl vytvořen algoritmus, který zautomatizoval tuto činnost. Ten fungoval na takovém principu, že sečetl náhodný signál ruchu s nahrávkou slova, který se poté uložil do nové složky jako *.wav* soubor a zařadil se pod správný adresář tvořený z názvů slov. Takto to bylo vykonáno pro všech 23 686 původních nahrávek. Než se signál ruchových nahrávek sečetl, musel být ještě před tím upraven. Jelikož nahrávky ruchů byly dlouhé, musely se zkrátit na stejnou délku, jaká byla délka sčítané nahrávky slova a zároveň, jelikož tyto ruchy byly příliš hlasité a zcela by přebily ve výsledné nahrávce vyslovené slovo, musely se tyto ruchy také ztišit.

Také bylo otestováno, jak by si NN vedla, kdyby byla trénována na nahrávkách, ve kterých by byl pomocí funkce redukován již zmíněný ruch. Tento dataset by nebyl stejný jako původní z důvodu nepřesnosti redukované funkce. Můžeme tedy předpokládat, že výsledná přesnost a ztráta modelu bude rozdílná.

5.1 Dataset původních nahrávek

Model natrénovaný na tomto datasetu byl částečně rozebrán v sekci 4.4 v předchozí kapitole. Model má dobrou přesnost (*Test accuracy*: 0,956) a relativně nízkou ztrátou (*Test loss*: 0,318). Ovšem když přijde na testování nahrávek, které jsou mimo tento dataset původních nahrávek, tedy nahrávky obsahují nějakou formu ruchů, nebo artefakty vzniklé z použití filtrů, model výrazně zaostává. Při zhodnocení nahrávek obsahující ruch je přesnost o dost nižší (*Test accuracy*: 0,872) a hodnota ztráty výrazně vyrostla (*Test loss*: 1,219). Obdobné výsledky dosahuje při testování i dataset, ve kterém byly pomocí externí funkce odstraněny ruchy (*Test loss*: 1,146; *Test accuracy*: 0,875). Všechny hodnoty lze najít v tabulce 5.1.

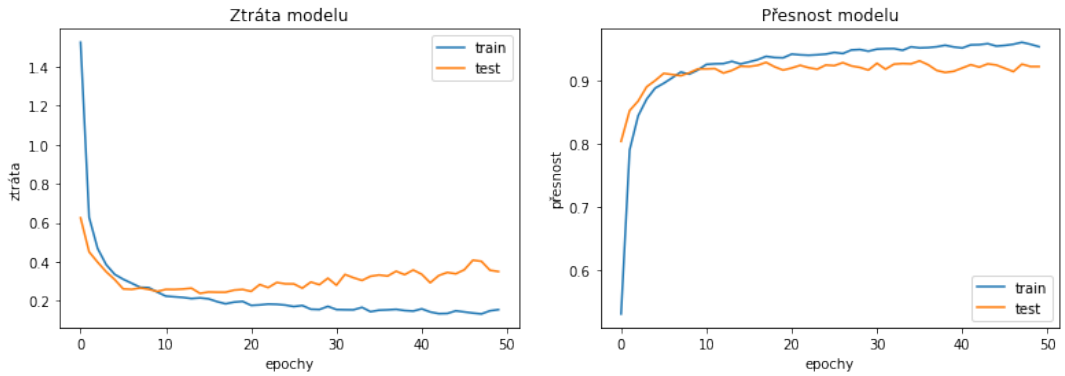
Tab. 5.1: Tabulka výkonosti různých typů testovacích nahrávek na modelu natrénovaném na čistých nahrávkách

Typ datasetu	*Test loss*	*Test accuracy*
Čisté nahrávky	0,318	0,956
Nahrávky obsahující ruch	1,219	0,872
Nahrávky s odstraněným ruchem	1,146	0,875

Toto chování je pravděpodobně z toho důvodu, že model si nedokáže poradit s novými informacemi obsahujících v nahrávkách. Při přidání ruchů do nahrávek přidáváme nové informace do oběhu. Jelikož pro neuronovou síť jsou tyto informace taktéž novější, mohlo se předpokládat, že přesnost modelu poklesne. Tuto tezi lze obdobně aplikovat i na dataset s odstraněnými ruchy. Sice novější informace jsou zde potlačeny (dalo by se předpokládat, že model by si vedl lépe), ovšem při použití externí funkce dochází ke vzniku umělých artefaktů v nahrávkách. S těmito artefakty si opět model nedokáže tak dobře poradit.

5.2 Dataset nahrávek s přidanými ruchy

Model byl natrénován na nahrávkách, které obsahovaly ruchové elementy. Model měl stejnou architekturu jako předchozí model, který byl natrénovaný na čistých nahrávkách. Byl trénovaný na stejný počet epoch, se stejnou velikostí `batch_size`. Všechno bylo stejné, kromě zmíněných trénovacích dat. Na obrázku 5.1 si můžeme všimnout, že ztráta tohoto modelu dosahuje obdobných hodnot, ale přesnost modelu je o trochu nižší než u předchozího modelu.



Obr. 5.1: Grafy změn hodnot ztráty a přesnosti modelu trénovaný na nahrávkách obsahující ruchy

Tab. 5.2: Tabulka výkonosti různých typů testovacích nahrávek na modelu natrénovaném na nahrávkách obsahující ruchy

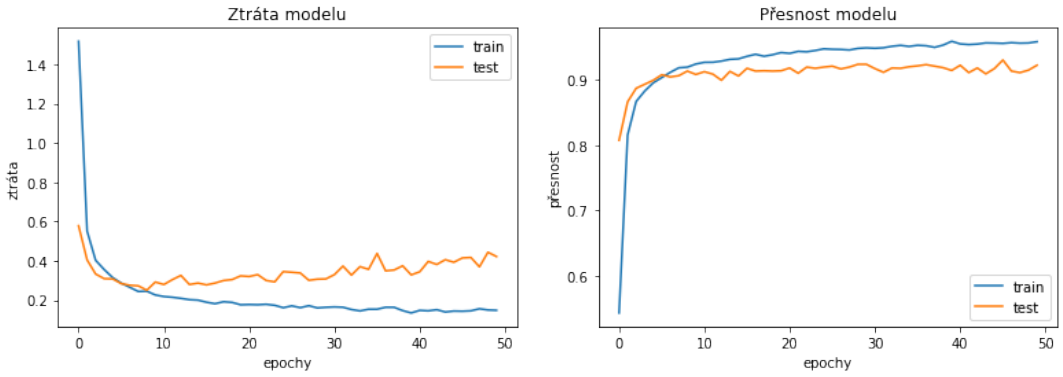
Typ datasetu	*Test loss*	*Test accuracy*
Čisté nahrávky	0,261	0,964
Nahrávky obsahující ruch	0,339	0,927
Nahrávky s odstraněným ruchem	0,393	0,922

Přesnost modelu, který byl natrénován na nahrávkách obsahující ruchy a byl zhodnocen na týž samých nahrávkách dosahovala hodnoty 0,927 a ztráta při tomto testování byla 0,339. Při testování na ostatních datasetech dosahuje obdobných výsledků. Při testování na čistých nahrávkách, model dosahuje přesnosti 0,964 a ztráty 0,261. Při testování na nahrávkách s odstraněnými ruchy dosahuje ovšem nižší přesnosti a razantně vyšší ztrátu (*Test loss*: 0,393; *Test accuracy*: 0,922), ale tento skok není tak razantní jako u předchozího modelu. Všechny hodnoty lze najít v tabulce 5.2.

Tento případ je přesným opakem modelu natrénovaného na čistých nahrávkách. Jelikož model zná všechny informace, které kdy obdrží, dokáže díky tomu konat mnohem přesněji. Proto se zde neobjevují tak razantní výkyvy hodnocení jako u předchozího modelu.

5.3 Dataset nahrávek s odstraněnými ruchy

Dataset byl vytvořen pomocí funkce pro odstranění šumu (více v sekci 6.2), která byla aplikována na nahrávky, jenž obsahovaly ruchový prvek. Model natrénovaný na tomto datasetu dosahuje obdobných hodnot jako model natrénovaný na nahrávkách



Obr. 5.2: Grafy změn hodnot ztráty a přesnosti modelu trénovaný na nahrávkách s odstraněnými ruchy

Tab. 5.3: Tabulka výkonosti různých typů testovacích nahrávek na modelu natrénovaném na nahrávkách s odstraněnými ruchy

Typ datasetu	*Test loss*	*Test accuracy*
Čisté nahrávky	0,255	0,951
Nahrávky obsahující ruch	0,219	0,941
Nahrávky s odstraněným ruchem	0,387	0,924

obsahující ruchy (*Test loss*: 0,387; *Test accuracy* 0,924). Model má taktéž obdobnou historii průběhu přesnosti a ztráty (obr. 5.2). Ovšem u testování ostatních datasetů si vede v průměru o trochu lépe než předchozí model.

Pro čisté nahrávky dosahuje model přesnosti 0,951 a ztráty 0,255. Tyto hodnoty jsou dosti obdobné jako hodnoty u předchozího modelu. Avšak zhodnocení nahrávek obsahující ruchy dosahuje značného zlepšení (*Test loss*: 0,219; *Test accuracy*: 0,941). Průměrná hodnota těchto přesností je sice téměř shodná, ale průměrná ztráta testování je nižší než u předchozího modelu (konkrétně o 0,044). Nejdůležitější je zde fakt, že hodnoty přesnosti jsou více rozprostřené mezi datasety. Dá se formulovat, že medián hodnot přesností je v tomto modelu vyšší než u modelu natrénovaném na nahrávkách obsahující ruchy. Všechny hodnoty lze najít v tabulce 5.3.

Toto chování je pravděpodobně způsobené tím, že při nahrávkách obsahující ruch model dokáže lépe rozlišit ruchovou část nahrávky a řečovou část nahrávky. Dataset, na kterém byl trénován neobsahoval tolik ruchu, ovšem stále obsahoval artefakty zanechané právě zmíněným ruchem. Toto je ovšem pouhá teze a tento argument není podložený bližším zkoumáním.

6 Implementace systému

Výsledný systém pro rozpoznání zvukových povelů obsahuje několik dílčích částí kromě samotného klasifikátoru. Blokové schéma tohoto systému lze najít v kapitole 3.1 (obr. 3.1).

6.1 Detekce hlasové aktivity

VAD bylo realizováno pomocí objektu `Vad` z knihovny `webrtcvad`. Tato funkce je založena na metodě, která byla vytvořena pro komunikaci přes internet v reálném čase. Jedná se o metodu energetického prahování.

Vytvoření objektu `Vad` umožňuje nastavení budoucího chování VAD. Pomocí metody `set_mode()` je možné VAD nastavit až do 4 různých režimů agresivity. Hodnota 0 značí nejméně agresivní režim a hodnota 3 nejvíce agresivní režim.

Metoda `is_speech()`, která je obsažena v tomto objektu, vrací boolean hodnotu, pokud se v úseku nachází řečový signál nebo ne. Do této metody lze poslat pouze jenom úsek, který je dlouhý 10, 20 nebo 30 milisekund. Další nevýhoda této metody je, že může zpracovat pouze úseky, jejichž vzorkovací frekvence je 8000, 16000, 32000 nebo 48000 Hz. Zároveň úsek musí být vytvořený pomocí pulzně kódované modulace (PCM) a mít 16bitové rozlišení.

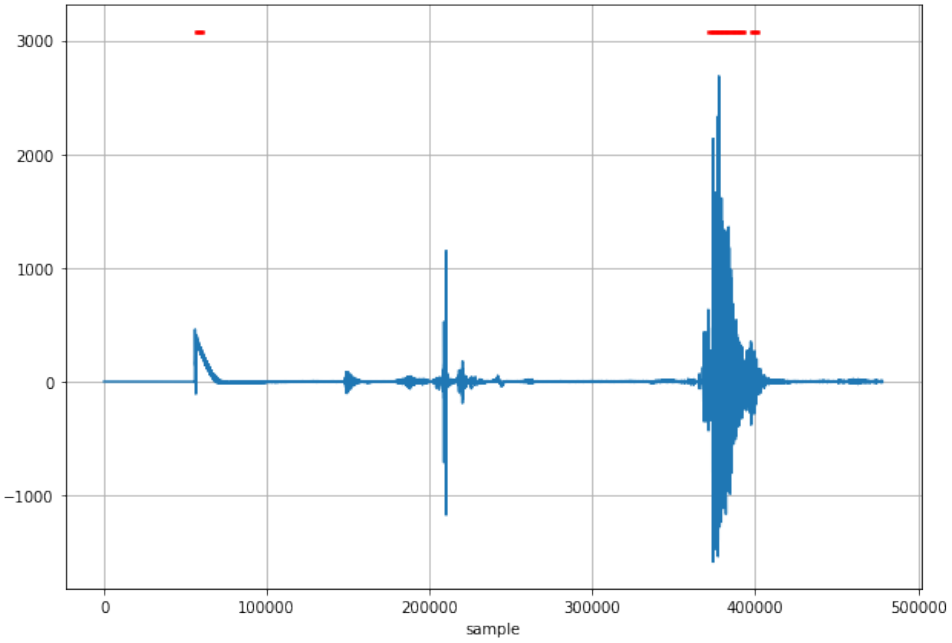
Nahrávka tedy musí být převzorkovaná, pokud její vzorkovací frekvence nedosahuje vypsaných hodnot. Zároveň nahrávka musí být rozkouskována na dílčí částí, které poté postupně projdou přes metodu `is_speech()` aby se zjistilo, ve kterých úsecích je obsažena mluva a ve kterých není. Musí být také zajištěno, že všechny nahrávky mají 16bitové rozlišení. Výsledné úseky, které byly vyhodnoceny jako úseky obsahující mluvu, můžou být poté vystřiženy a uloženy zvlášť do nahrávky. Také je příhodné využít nějaký typ filtru pro vyčištění chybně zhodnocených úseků.

Převzorkování nahrávky lze dosáhnout pomocí již zmíněné funkce `load`, která je obsažena v knihovně `librosa`. Jak již bylo řečeno, funkce `load` automaticky převzorkovává vybranou nahrávku na 22 050 Hz. Ovšem tuto hodnotu lze změnit.

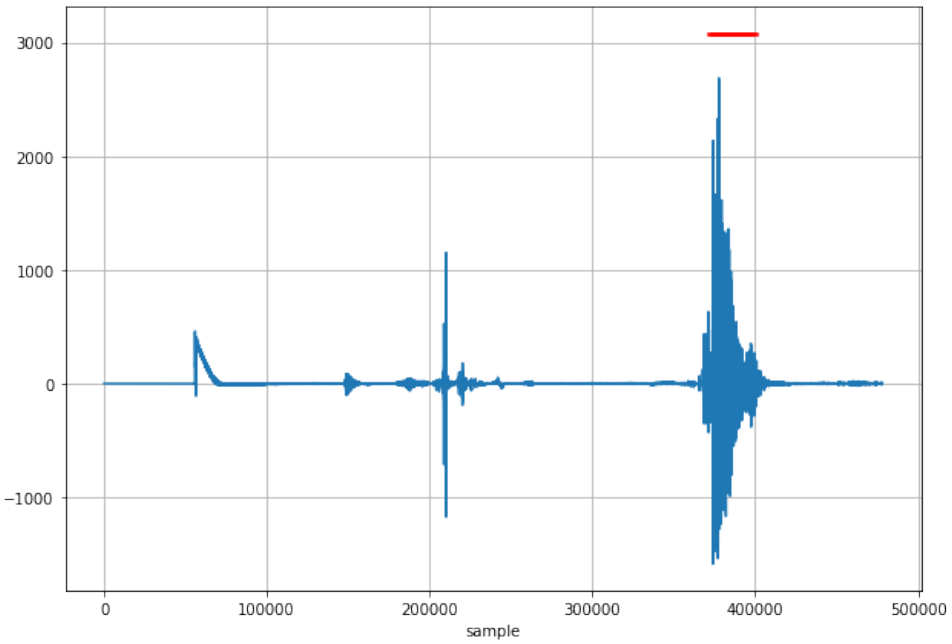
Zajištění nahrávky, aby měla 16bitové rozlišení, zajišťuje funkce `pack` z knihovny `struct`. Ta dokáže převést celočíselné hodnoty (také hodnoty s pohyblivou řádovou čárkou ovšem s jiným nastavením) do 16bitového řetězce hodnot.

Výsledný algoritmus (převzato z [18]) uloží do slovníku všechny úseky a roztřídí je pod klíčové hodnoty na základě výsledku VAD. Na obrázku 6.1 se nachází zvýrazněné úseky (červené úsečky), které VAD vyhodnotilo, že obsahují řečový signál. Testovací nahrávka obsahuje v prvních dvou třetinách 2 různé typy ruchů a mluvené slovo se objevuje až ke konci ve třetí třetině nahrávky. Můžeme si zde povšimnout,

že VAD mylně vyhodnotilo první ruch v nahrávce a označilo ho za úsek obsahující řeč, a zároveň mylně vyhodnotilo pár úseků v části obsahující řeč.



Obr. 6.1: Výstup VAD na testovací nahrávce



Obr. 6.2: Výstup VAD s aplikovaným filtrem na testovací nahrávce

Tento problém lze vyřešit pomocí aplikace filtru výsledky zhodnocení VAD. Filtr pracuje obdobně jako mediánový filtr, avšak je postavený na rozdílnějším principu

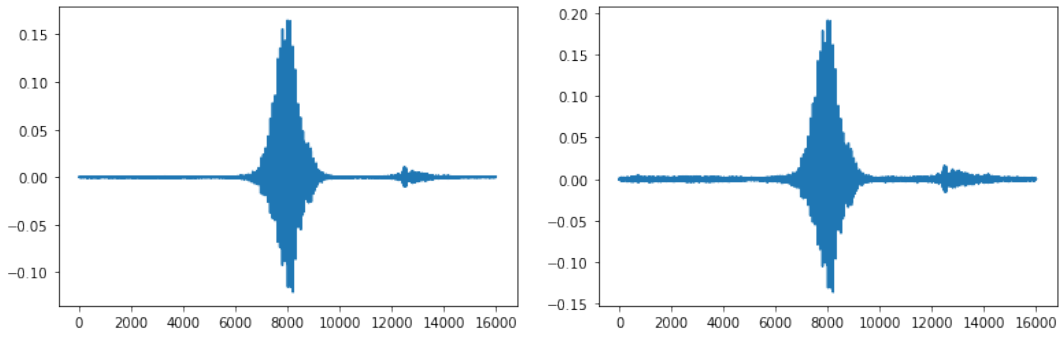
fungování. Filtr odstraní mylně vyhodnocené úseky a doplní úseky v místech obsahující řečový signál. Úseky, které byly mylně vyhodnoceny, nesmí překračovat celkovou délku úseků 150 milisekund. Vyhodnocení testovací nahrávky pomocí VAD po aplikaci filtru lze najít na obrázku 6.2.

Výsledný úsek byl poté vystřižen pomocí funkce `concatenate` z knihovny `numpy` a následně uložen zvlášť do zvukového souboru pod názvem `vad.wav` pro možnou budoucí kontrolu a případnou manipulaci.

6.2 Odstranění šumu

Odstranění šumu z nahrávek bylo realizováno pomocí funkce `reduce_noise` z obdobně jmenné knihovny `noisereduce`. Tato funkce funguje na principu spektrální brány (Spectral Gate), která je formou šumové brány (Noise Gate).

V této funkci lze nastavit řadu parametrů, ovšem nejdůležitějším parametrem, kromě tedy vyžadovaných parametrů, které jsou samotný signál a vzorkovací frekvence, je parametr `stationary`. Ten rozhoduje, zda odstranění ruchu pracuje na principu stacionární, nebo nestacionární redukce šumu.

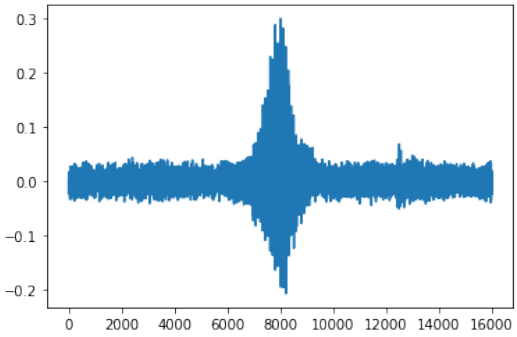


(a) (b)

Obr. 6.3: (a) Stacionární redukce, (b) Nestacionární redukce

Obrázek 6.3 značí to, že stacionární redukce šumu redukuje šum více než nestacionární. To je pravda, ovšem stacionární redukce redukuje i důležité prvky řečového signálu. Sice šum zredukuje více, ale razantněji ovlivní výslednou kvalitu nahrávky. Proto byl zvolen nestacionární model, který ruchy nezredukuje tolik, ale zanechá kvalitu řečového signálu. Původní nahrávka s ruchem se nachází na obrázku 6.4.

Tato funkce byla použita pro automatizaci výroby nahrávek, ve kterých byly odstraněny ruchy. Algoritmus fungoval obdobně jako algoritmus pro tvorbu nahrávek obsahující šum. Jediný rozdíl byl ten, že místo sčítání čisté nahrávky a šumu, se vzala nahrávka obsahující ruch a aplikovala se na ní funkce `reduce_noise` v nestacionárním režimu. Výsledná nahrávka se opět uložila do příslušného adresáře.

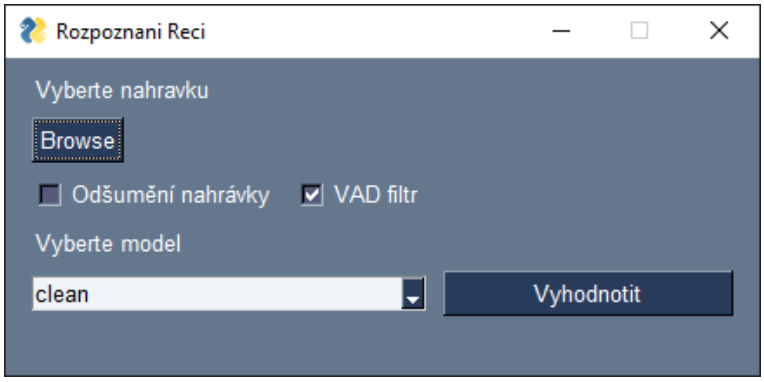


Obr. 6.4: Původní nahrávka pro demonstraci šumové redukce

Taktéž tato funkce byla přidána jako dobrovolná funkce, která se v systému aplikuje na samotnou vstupní nahrávku ještě před tím, než ji zpracuje VAD. Zapnutí této funkce lze v uživatelském rozhraní.

6.3 Uživatelské rozhraní

Pro lepší orientaci mezi jednotlivými dílčími částmi obsahující v systému, bylo vytvořeno jednoduché uživatelské rozhraní, které ulehčí uživateli ovládání celkového systému. Toto rozhraní bylo vytvořeno pomocí knihovny `PySimpleGUI`.

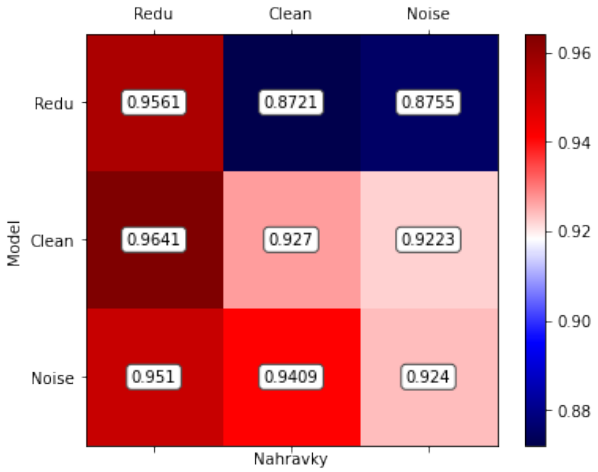


Obr. 6.5: Uživatelské rozhraní systému

Rozhraní je vyobrazeno na obrázku 6.5 a skládá ze 4 hlavních částí:

1. Vybrání nahrávky pro klasifikaci,
2. nastavení možnosti externího zpracování,
3. vybrání modelu pro klasifikaci,
4. okno pro výsledek samotné klasifikace.

Toto rozhraní spojuje všechny části obsažené v této práci. Nejprve VAD vysekne z vybrané nahrávky část, která obsahuje řečový signál. Následně tato část upraví



Obr. 6.6: Závislost přesnosti modelů na typech nahrávkách

svoje rozměry a převede se na MFCC. Ty poté putují do jednoho z vybraných natrénovaných modelů neuronové sítě, kde se klasifikují. Na závěr se systém podívá na index nejvyšší hodnoty z pole, které klasifikátor vyhodnotil, a přiřadí jeho příslušný povel, který se následně zobrazí v dolní části uživatelského rozhraní.

Možné modely neuronové sítě jsou modely, které byly trénované v předchozí kapitole, tedy model natrénovaný na čistých nahrávkách, model natrénovaný na nahrávkách obsahující ruch a model natrénovaný na nahrávkách s odstraněnými ruchy. Přesnost těchto modelů v závislosti na jednotlivých typech nahrávkách lze najít v jednotlivých sekcích zabývajících se danými modely nebo na obrázku 6.6.

Závěr

Hlavním úkolem této bakalářské práce bylo vytvoření systému pro rozpoznání hlasových povelů. Tento systém byl realizován v jazyce python a systémový klasifikátor byl vytvořený pomocí konvoluční neuronové sítě.

Kromě historie a samotného vysvětlení používaných metod pro rozpoznání hlasových povelů, byly v teoretické části vysvětleny principy a realizace detekce hlasové aktivity a odstranění šumů ze signálu.

V praktické části byl vypsány hlavní části, ze kterých se systém pro rozpoznání hlasových povelů nejčastěji skládá. Byl zde i popsán návrh samotné neuronové sítě, která byla použita jako klasifikátor. Jelikož výsledkem neuronové sítě je pole hodnot, byla zde vypsaná důležitá tabulka obsahující jednotlivé povely a jejich příslušné indexy pro budoucí klasifikaci nahrávek.

Pro vytvoření datasetu z nahrávek, které byly využity z volně dostupné databáze od společnosti Google, byly využité mel-frekvenční cepstrální koeficienty. Ovšem aby všechny nahrávky převedené na MFCC měly stejné rozměry, musely být nahrávky předtím ještě unifikovány.

Pomocí knihovny `keras-tuner` bylo vytvořených přes 300 unikátních neuronových sítí ve snaze najít tu, která dosahuje nejpřesnějšího rozhodování. Z výsledné architektury neuronové sítě byly vytvořeny 3 modely, které byly natrénovány na 3 různých datasetech. Mezi tyto datasety patří dataset obsahující čisté původní nahrávky, dataset obsahující nahrávky s přidanými ruchy a dataset obsahující nahrávky s odstraněnými ruchy pomocí externí funkce.

Všechny tyto modely byly testovány na ostatních datasetech. Model, který měl nejpřesnější výsledek, byl natrénovaný na datasetu s nahrávkami obsahující ruchy. Tento model při testování na čistých původních nahrávkách dosahoval přesnosti 96,4 %. Tento model si i relativně dobře vedl na testování ostatních datasetů. Je to pravděpodobně způsobeno tím, že model je natrénovaný na nahrávkách obsahující ruchy, to znamená, že zná všechny možnosti, které může model obdržet pro zhodnocení. Přesným opakem je model natrénovaný na původních čistých nahrávkách. Ten při testování na čistých nahrávkách dosahoval 95,6 % ovšem při testování ostatních datasetů výrazně ztrácel. Jelikož tento model byl trénovaný pouze na čistých nahrávkách (tedy nahrávkách, které obsahovaly buď žádnou, nebo minimální ruchovou složku), nedokázal se vypořádat s informacemi se kterými se nikdy nesetkal. Ovšem nejlepším modelem v průměru mezi vším testováním byl model, který byl natrénovaný na nahrávkách s odstraněnými ruchy.

Byl vytvořený VAD algoritmus. Ten fungoval na principu, že nejprve odstranil ruchy z nahrávky. Poté se v nahrávce detekovaly úseky obsahující řečový signál a následně na tyto úseky byl aplikován filtr, který se postaral o mylné detekování

úseků, které neobsahovalo mluvu.

Všechny tyto dílčí části byly sjednoceny pomocí jednoduchého uživatelského rozhraní. Uživatel si může zvolit v jaké nahrávce by chtěl detekovat slovo. Tato nahrávka poté putuje do VAD algoritmu, ve kterým si ovšem uživatel může nastavit, zda-li chce aplikovat odstranění šumu na nahrávku, případný čistící filtr. Uživatel si také může zvolit jeden ze 3 natrénovaných modelů.

V práci je možné pokračovat. Jedním ze způsobů by bylo rozšíření databáze slov, které by systém dokázal detekovat. Také by bylo možné rozšířit uživatelské rozhraní o spoustu dodatečného nastavení pro naprostou kontrolu nad systémem.

Literatura

[1] Hill, Paul *Audio and speech processing with MATLAB.* Boca Raton: CRC Press/Taylor & Francis Group, 2018. ISBN 978-0-429-44406-7.

[2] Clynn *FFT-Time-Frequency-View-540.png.* Dostupné z URL: <https://classes.engineering.wustl.edu/ese205/core/index.php?title=File:FFT-Time-Frequency-View-540.png>.

[3] Van Loan, Charles *Computational Frameworks for the Fast Fourier Transform.*, SIAM, 1992

[4] Sarkar, Priya *Speech Analytics Part -1, Basics of Speech Analytics* 16. 8. 2020 Dostupné z URL: <https://medium.com/analytics-vidhya/speech-analytics-part-1-basics-of-speech-analytics-37ba6d5904e2>.

[5] Min, Xu; *HMM-based audio keyword generation*(PDF). et al. (2004). In Kiyoharu Aizawa; Yuichi Nakamura; Shin'ichi Satoh (eds.). Advances in Multimedia Information Processing – PCM 2004: 5th Pacific Rim Conference on Multimedia. Springer. ISBN 978-3-540-23985-7. Archivováno z pdf 10. 5. 2007.

[6] Gündert, Siegfried *melbank-1_00.hires.* 2014 Dostupné z URL: <https://siggigue.github.io/pyfilterbank//melbank-1_00.hires.png>.

[7] Tavenard, Romain *An introduction to Dynamic Time Warping.* Dostupné z URL: <https://rtavenar.github.io/blog/fig/dtw_vs_euc.svg>.

[8] Gratz, Jaroslav *Jazykové modely pro vyhledávání: naučte stroj chápat význam jazyka*, 8. 9. 2021 Dostupné z URL:

<https://www.root.cz/clanky/jazykove-modely-pro-vyhledavani-naucte-stroj-chapat-vyznam-jazyka>

[9] Domingos, Pedro. *The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World*, Basic Books 22. 9. 2015 ISBN 978-0-465-06570-7.

[10] Russell, Stuart Jonathan; Norvig, Peter *Artificial Intelligence: A Modern Approach* (Third ed.). 2010 Prentice Hall. ISBN 978-0-136-04259-4.

[11] van Otterlo, Martijn; Wiering, Marco *Reinforcement learning and markov decision processes. Reinforcement Learning. Adaptation, Learning, and Optimization.* 2012 doi:10.1007/978-3-642-27645-3_1. ISBN 978-3-642-27644-6.

[12] Gomez, Christian *WHAT IS MACHINE LEARNING?* Dostupné z URL: <https://medium.com/@1154_75881/what-is-machine-learning-fb3821129ff0>

[13] Hopfield, John Joseph *Neural networks and physical systems with emergent collective computational abilities.* 1982 Proc. Natl. Acad. Sci. U.S.A. 79 (8). Bibcode:1982PNAS...79.2554H. doi:10.1073/pnas.79.8.2554. PMC 346238. PMID 6953413.

[14] Ramachandran, Ravi; Mammone, Richard *Modern Methods of Speech Processing.* (6 December 2012) Springer Science and Business Media. ISBN 978-1-4615-2281-2.

[15] Boashash, B *Time-Frequency Signal Analysis and Processing – A Comprehensive Reference.* ed. (2003). Oxford: Elsevier Science. ISBN 978-0-08-044335-5.

[16] O'Malley, Tom and Bursztein, Elie and Long, James and Chollet, François and Jin, Haifeng and Invernizzi, Luca and others *KerasTuner* 2019 Dostupné z URL: <https://github.com/keras-team/keras-tuner>

[17] Bäuerle, Alex; van Onzenoodt, Christian;Ropinski, Timo *Net2Vis – A Visual Grammar for Automatically Generating Publication-Tailored CNN Architecture Visualizations* 8. 2. 2021 Publikováno v IEEE Transactions on Visualization and Computer Graphics (Volume: 27, Issue: 6, June 1 2021)

[18] Holzner, Andre *voice activity detection example* Dostupné z URL: <https://www.kaggle.com/code/holzner/voice-activity-detection-example/notebook>

Seznam symbolů a zkratek

HMM	Skryté Markovovy modely – Hidden Markov model
FFT	Rychlá Fourierova transformace – Fast Fourier transform
DFT	Diskrétní Fourierova transformace – Discrete Fourier transform
MFC	Mel-frekevnční cepstrum – Mel-frequency cepstrum
MFCC	Mel-frekevnční cepstrální koeficienty – Mel-frequency cepstral coefficients
DTW	Dynamická časová deformace – Dynamic time warping
ML	Strojové učení – Machine learning
SL	Učení s učitelem – Supervised learning
RL	Zpětnovazební učení – Reinforcement learning
MDP	Markovův rozhodovací model – Markov decision process
AI	Umělá inteligence – Artificial intelligence
NN	Neuronové sítě – Neural networks
ANN	Umělé neuronové sítě – Artificial neural networks
ANN	Umělé neuronové sítě – Artificial neural networks
P	Perceptron – Perceptron
MP	Vícevrstvý perceptron – Multilayer perceptron
FF	Neuronová síť s dobřednou vazbou – Feed Forward Neural Networks
RBF	Neuronová síť s radiální bází – Radial Basis Function Neural Networks
MNN	Modulární neuronová síť – Modular Neural Network
RNN	Rekurentní neuronová síť – Recurrent Neural Networks
CNN	Konvoluční neuronová síť – Convolution Neural Networks
VAD	Detekce hlasové aktivity – Voice Activity Detection

A Obsah příloh

Nahrávky obsažené v příloze nepokrývají rozsáhlost všech nahrávek použitých v této práci. Byla vyjmuta pouhá část nahrávek pro stálou možnou demonstranci testování neuronové sítě.

```
Bakalarska_prace
├── nahravky ..... Složka obsahující čisté nahrávky
│   ├── Down
│   │   ├── Down (1).wav
│   │   ├── Down (2).wav
│   │   └── ...
│   ├── Left
│   │   ├── Left (1).wav
│   │   ├── Left (2).wav
│   │   └── ...
│   ├── No
│   │   ├── No (1).wav
│   │   ├── No (2).wav
│   │   └── ...
│   └── ...
├── clean16_512 ..... Model NN natrénovaný na čistých nahrávkách
├── noise16_512 ..... Model NN natrénovaný na nahrávkách obsahující ruchy
├── redu16_512 ..... Model NN natrénovaný na nahrávkách s odstraněnými ruchy
├── dataset.py ..... Vytvoření datasetu z MFCC
├── gui.py ..... Uživatelské rozhraní
├── main.ipynb ..... Konvoluční neuronová síť
├── MFCC.py ..... Převedení audio nahrávek na MFCC
├── noiseset.py ..... Skript pro automatizaci tvorby zašumělých nahrávek
├── reduset.py ..... Skript pro automatizaci tvorby nahrávek s odstraněným ruchem
├── tuning.py ..... Skript pro ladění neuronové sítě
├── vadtest.py ..... Podpůrná knihovna pro GUI
└── xsible00.pdf ..... Bakalářská práce ve formě pdf
```